EXHIBIT H

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

Exhibit C-10

Based on Headwater's apparent positions as to the scope of the patent's claims, as best they can be deciphered, the reference(s) charted below anticipate(s) or at least render(s) obvious the identified claims. The portions of the prior art system documents cited below are not exhaustive but are exemplary in nature. Additional citations may be found in the cover pleading.

This disclosure is not an admission that Samsung concedes any claim construction implied or suggested by Headwater's apparent positions as to the scope of the patent's claims, nor is it an admission by Samsung that any of its products are covered by or infringe the patent's claims, particularly when they are properly construed and applied. Samsung is not taking any claim construction positions through this disclosure, including whether the preamble is a limitation.

Samsung reserves the right to rely on additional citations or sources of evidence that also may be applicable, or that may become applicable in light of claim construction, changes in Headwater's infringement contentions, and/or information obtained during discovery as the case progresses. Samsung further reserves the right to amend or supplement this claim chart at a later date as more fully set forth in the Invalidity Contentions. For example, Defendants are currently in the process of taking discovery from non-parties including Nokia, HMD, Citrix, Google, Apple, and Microsoft. Accordingly, Defendants reserve the right to modify, amend, and/or supplement these contentions as information becomes available from non-parties.

Android is mobile device operating system that was initially released in September 2008. Applications (or "apps") can be installed on mobile devices that run Android. Any mobile device that predates the '578 patent, running an Android version with one or more apps that also predate the '578 patent, qualifies as prior art under at least pre-AIA 35 U.S.C. §§ 102(a)/(b). Such a device was known, used, offered for sale, and/or sold in the United States before the '578 patent.

Exemplary mobile devices that predate the '578 patent and were publicly available before the earliest possible priority date include: [1]

- HTC Dream/T-Mobile G1 (released September 2008)
- Samsung GT-I7500 Galaxy (released June 2009)

[1] *See, e.g.*, SAMSUNG_PRIORART0000001-334; SAMSUNG_PRIORART0005174-76; SAMSUNG_PRIORART0005177-317; SAMSUNG_PRIORART0005416-19; SAMSUNG_PRIORART0005420-23; SAMSUNG_PRIORART0005424-28; SAMSUNG_PRIORART0005429-44; SAMSUNG_PRIORART0005445-48; SAMSUNG_PRIORART0005449-52; SAMSUNG_PRIORART0005453-57; SAMSUNG_PRIORART0005458-71; SAMSUNG_PRIORART0005472-77; SAMSUNG_PRIORART0005478-84; SAMSUNG_PRIORART0005485-86; SAMSUNG_PRIORART0005488-5624.

- Nexus One (released January 2010)

Exemplary Android versions that predate the '578 patent and were publicly available before the earliest possible priority date include: [2]

- Android 1.0 (released September 2008)
- Android 1.1 (released February 2009)
- Android Cupcake (1.5) (released April 2009)
- Android Donut (1.6) (released September 2009)
- Android Eclair (2.0, 2.0.1, 2.1) (released October 2009 - January 2010)
- Android Froyo (2.2) (released May 20, 2010)

Android included files[3] such as ConnectivityManager, NetworkInfo, NetworkStateTracker, ThrottleManager, TrafficStats, ConnectivityManagerMobileTest, Socket, SocketTest, Power, PowerManager, PowerManagerTest, BatteryManager, and BatteryStats.

Exemplary apps that predate the '578 patent and were publicly available before the earliest possible priority date include: [4]

---

[2] See, e.g., SAMSUNG_PRIORART0003998; SAMSUNG_PRIORART0004085; SAMSUNG_PRIORART0004081; SAMSUNG_PRIORART0004086; SAMSUNG_PRIORART0004083; SAMSUNG_PRIORART0004084.
[3] The files listed are Java source files, so the filenames are, e.g., ConnectivityManager.java, NetworkInfo.java, etc., except as noted.
[4] See, e.g., SAMSUNG_PRIORART0000335-SAMSUNG_PRIORART0000383; POUZERATE0000001-POUZERATE0000261 GOOG-HEADWATER-000000001-123; HDWTR-GOOG00001-GOOG00013; SAMSUNG_PRIORART0005042; SAMSUNG_PRIORART0005062; SAMSUNG_PRIORART0005350; SAMSUNG_PRIORART0005351; SAMSUNG_PRIORART0005352; SAMSUNG_PRIORART0005353; SAMSUNG_PRIORART0005354; SAMSUNG_PRIORART0005355; SAMSUNG_PRIORART0005356; SAMSUNG_PRIORART0005357; SAMSUNG_PRIORART0005358; SAMSUNG_PRIORART0005359; SAMSUNG_PRIORART0005360; SAMSUNG_PRIORART0005361; SAMSUNG_PRIORART0005362; SAMSUNG_PRIORART0005363; SAMSUNG_PRIORART0005364; SAMSUNG_PRIORART0005046; SAMSUNG_PRIORART0005043; SAMSUNG_PRIORART0005044; SAMSUNG_PRIORART0005045; SAMSUNG_PRIORART0005054; SAMSUNG_PRIORART0005055; SAMSUNG_PRIORART0005056; SAMSUNG_PRIORART0005057;

- JuiceDefender (released January 2010) and its associated add-on application, UltimateJuice (collectively "JuiceDefender App")
- GreenPower (released March 2010)

As specific examples, an HTC Dream/T-Mobile G1, Samsung GT-I7500 Galaxy, or a Nexus One mobile device running any of Android versions 1.0- 2.2 by itself, or with the JuiceDefender or GreenPower applications installed qualifies as prior art under at least pre-AIA 35 U.S.C. §§ 102(a)/(b). This device was known, used, offered for sale, and/or sold in the United States on or before May 20, 2010. At least the various documents cited in this claim chart describe the functionality of this device.

To the extent it is argued that Android Device with One or More Apps does not disclose or include each and every asserted claim limitation, either expressly or inherently, it would have been obvious to a POSITA to incorporate any of the teachings from the references identified in Exhibits C-01 through C-11, and C-C (whose exemplary citations for each limitation are incorporated herein) into Android Device with One or More Apps. Indeed, it would have been obvious to make such combinations and a POSITA would have had reason and motivation to make such combinations at least for reasons described herein and in the cover pleading.

'578 Claims	**Android Device with One or More Apps**
[1 pre] A wireless end-user device, comprising:	To the extent the preamble is a limitation, Android Device with One or More Apps discloses and/or renders obvious this element. For example, see the following passages and/or figures, as well as related disclosures:

SAMSUNG_PRIORART0005058; SAMSUNG_PRIORART0005059; SAMSUNG_PRIORART0005060; SAMSUNG_PRIORART0005061; SAMSUNG_PRIORART0005318; SAMSUNG_PRIORART0005398; SAMSUNG_PRIORART0005047-53; SAMSUNG_PRIORART0005063-65; SAMSUNG_PRIORART0005066-74; SAMSUNG_PRIORART0005075-135; SAMSUNG_PRIORART0005136-52; SAMSUNG_PRIORART0005153-70; SAMSUNG_PRIORART0005171-73; SAMSUNG_PRIORART0005319-49; SAMSUNG_PRIORART0005365-74; SAMSUNG_PRIORART0005375-83; SAMSUNG_PRIORART0005384-94; SAMSUNG_PRIORART0005395-97; SAMSUNG_PRIORART0005487.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

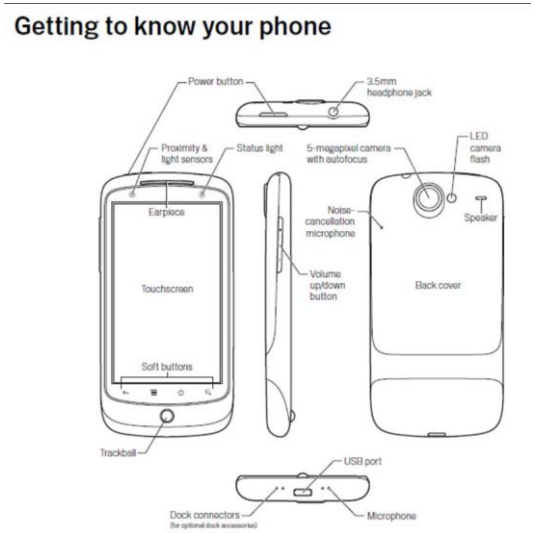
'578 Claims	Android Device with One or More Apps
	Nexus One The Nexus One is an example of an Android smartphone. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 17:  **HTC Dream / T-Mobile G1** The HTC Dream / T-Mobile G1 is an example of an Android smartphone. SAMSUNG_PRIORART0005184

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

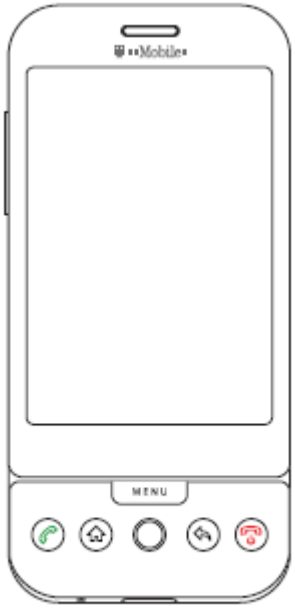
'578 Claims	Android Device with One or More Apps
	 **T-Mobile G1** SAMSUNG_PRIORART0005177 Android™ mobile technology platform R1.0 Document Rev 08 - September 8, 2008 Copyright 2008 © Google, Inc. All rights reserved. **Samsung GT-I7500 Galaxy**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	The Samsung GT-I7500 Galaxy is an example of an Android smartphone. SAMSUNG_PRIORART0005494 at pgs. PDF 2, 19: **<u>JuiceDefender App</u>** JuiceDefender is a mobile application (or "app") intended to run on a mobile device, such an Android smartphone. The Nexus One is an example of an Android smartphone capable of running JuiceDefender.

'578 Claims	Android Device with One or More Apps
	SAMSUNG_PRIORART0000379 (Latedroid) ("JuiceDefender saves battery power (lots of it!) by controlling the device data connection and/or WiFi … You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions.") SAMSUNG_PRIORART0000361 (Purdy) ("Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power.") **GreenPower App** GreenPower is a mobile application (or "app") intended to run on a mobile device, such an Android smartphone. The Nexus One is an example of an Android smartphone capable of running GreenPower. *See, e.g.*, POUZERATE0000015 (GDG Oslo) at 5:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	 See, e.g., POUZERATE0000002 (App Circus) at 9:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

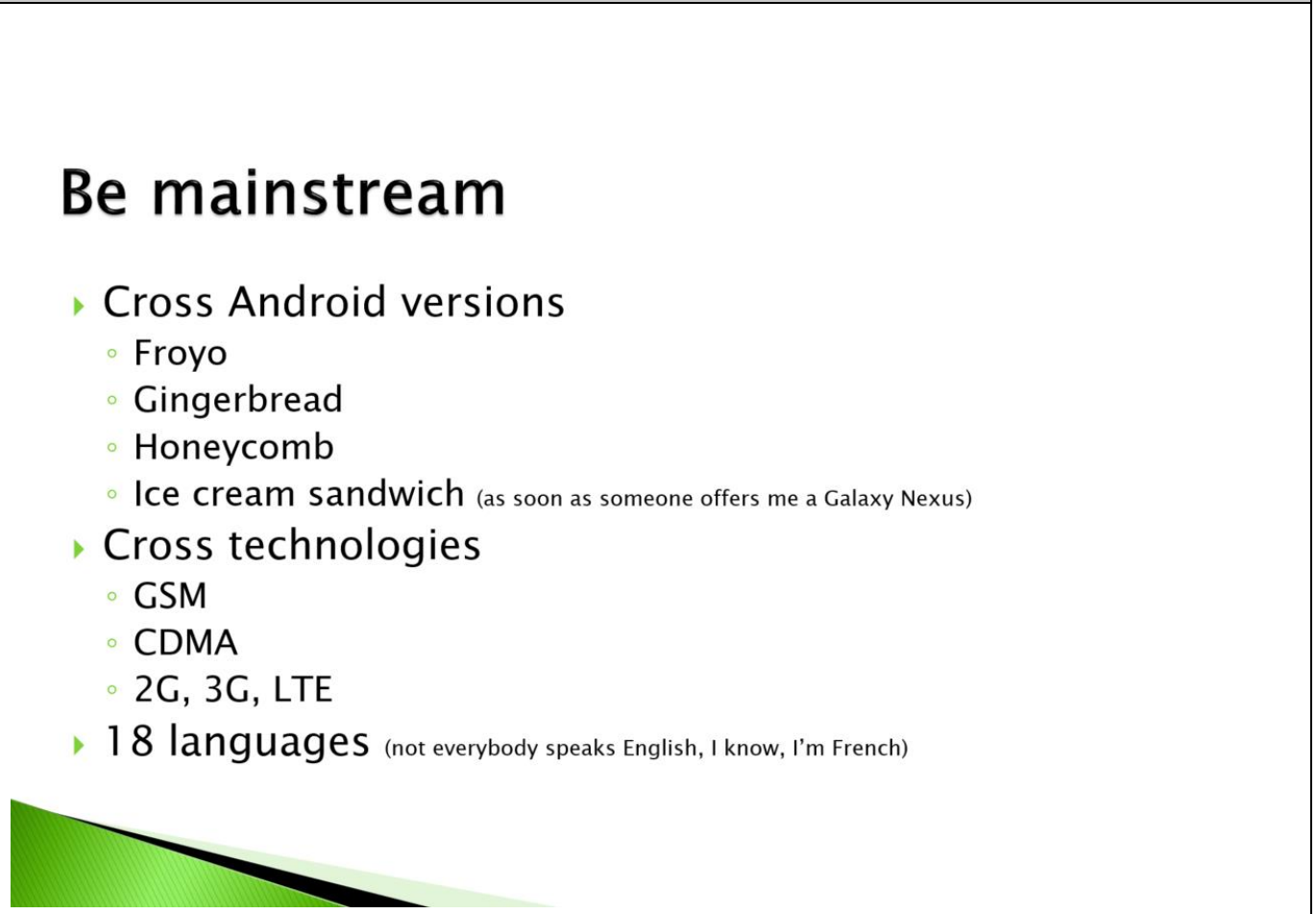
'578 Claims	Android Device with One or More Apps
	
[1a] a wireless wide area network (WWAN) modem to communicate data for Internet service	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: **Nexus One**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

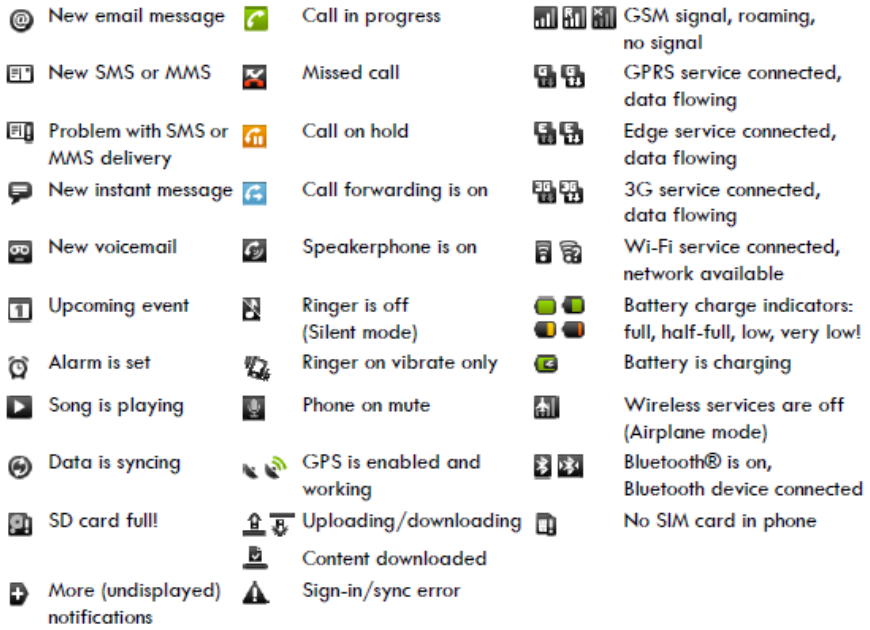
'578 Claims	Android Device with One or More Apps
activities between the device and at least one WWAN, when configured for and connected to the at least one WWAN;	*See, e.g.,* SAMSUNG_PRIORART0000001 (Nexus) at 332: Cellular & wireless Nexus One GSM phones compatible with 3G mobile networks from AT&T (U.S.) and Rogers Wireless (Canada): 3G UMTS bands I/II/V: 2100, 1900, 850 MHz Nexus One GSM phones compatible with 3G mobile networks from T-Mobile (U.S.): 3G UMTS bands I/IV/VIII: 2100, 1700(AWS), 900 MHz All Nexus One GSM phones: HSDPA 7.2Mbps HSUPA 2Mbps GSM/EDGE 850, 900, 1800, 1900 MHz Wi-Fi 802.11b/g Bluetooth 2.1 + EDR A2DP stereo Bluetooth **HTC Dream / T-Mobile G1** SAMSUNG_PRIORART0005202 **Mobile network settings** • Select data roaming capability. • Select to connect only to 2G (slower) networks to save battery power. • Select a wireless operator network - Scan for all available networks, or select a network automatically. • Add or edit network Access Point Names (APNs) - Do not change this setting unless advised to do so by your wireless operator! SAMSUNG_PRIORART0005200

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

Notification and connection status icons

Along the top of your phone screen is the status bar. On the left side, icons will appear, notifying you of a new message, upcoming calendar event, alarm, or something else you should notice. On the right side of the status bar, you'll see connection status icons.



Samsung GT-I7500 Galaxy

SAMSUNG_PRIORART0005494 at pg. PDF 21:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

Icons

Learn about the icons that appear on your screen.

Icon	Definition
	Signal strength
	GPRS network connected
	EDGE network connected
	UMTS network connected
	Roaming (outside of normal service area)
	GPS activated
	More status icons are available (touch the icon to see them)
	Call in progress
	Missed call

Icon	Definition
	Call diverting activated
	Connected to PC
	Bluetooth activated
	Bluetooth device connected
	Wi-Fi activated
	Synchronised with the web
	No SIM card
	New text message (SMS) or multimedia message (MMS)
	New email message
	New voice mail message
	Instant message

JuiceDefender App

SAMSUNG_PRIORART0000379 (Latedroid) ("JuiceDefender saves battery power (lots of it!) by controlling the device data connection and/or WiFi … You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions.").

SAMSUNG_PRIORART0000351 (Configuration-Translated) ("**APN:** activates / deactivates the APN connection, in its submenu we find the **MMS** button that activated configures the reception of MMS in the same way that we have the APN and **Prefer Wifi** that activated will try to connect first to this and if to five seconds does not find an available network will activate the APN. **WIFI:** activates / deactivates the WIFI connection, in its submenu we find the following buttons, **Auto Disable** turns off the wifi in the case of not finding a network available to save battery, in case it is deactivated we will have to activate it

'578 Claims	Android Device with One or More Apps
	manually; **Enable on Schedule / Peak / Screen will** activate the wifi as we have configured those buttons that we will see below.") SAMSUNG_PRIORART0000361 (Purdy) ("Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power."). SAMSUNG_PRIORART0000335 (Ruddock) ("Juice Defender is a battery conservation app. It uses various triggers, rules, and timers to control how often your device utilizes 3G/EDGE APN's (data connections) as well as WiFi. These data connections are the number one drainers of battery life when your phone is idle, so Juice Defender allows you to decide when, where, and how often you want them to be active."). **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Manage Mobile Network If this setting is selected, then Green Power will regularly turn on and off the Mobile Network connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Mobile Network as it is, never turning it on or off. Please note that in order for Green Power to turn on / off Mobile Network, this one has to be manually enabled by the user first in the phone settings (Wireless & networks → Mobile Network) or in Green Power settings (Global wireless settings → Mobile Network) . Green Power can't itself turn on Mobile Network as this is a limitation of the Android system for security and cost reasons."). **Android 1.0** **GOOG-HEADWATER-00000040, SAMSUNG_PRIORART0005487, ConnectivityManager**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* Class that answers queries about the state of network connectivity. It also
	* notifies applications when network connectivity changes. Get an instance
	* of this class by calling
	* {@link android.content.Context#getSystemService(String)
	Context.getSystemService(Context.CONNECTIVITY_SERVICE)}.
	* <p>
	* The primary responsibilities of this class are to:
	*
	* - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
	* - Send broadcast intents when network connectivity changes
	* - Attempt to "fail over" to another network when connectivity to a network
	* is lost
	* - Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	/** {@hide} */
	public boolean setRadios(boolean turnOn) {
	try {
	return mService.setRadios(turnOn);

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	} catch (RemoteException e) {
	return false;
	}
	}
	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {
	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {
	return false;
	}
	}
	
	SAMSUNG_PRIORART0005487, NetworkInfo
	/**
	* Indicates whether network connectivity exists or is in the process
	* of being established. This is good for applications that need to
	* do anything related to the network other than read or write data.
	* For the latter, call {@link #isConnected()} instead, which guarantees
	* that the network is fully usable.
	* @return {@code true} if network connectivity exists or is in the process
	* of being established, {@code false} otherwise.
	*/
	public boolean isConnectedOrConnecting() {
	return mState == State.CONNECTED || mState == State.CONNECTING;
	}
	
	/**
	* Indicates whether network connectivity exists and it is possible to establish

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* connections and pass data.
	* @return {@code true} if network connectivity exists, {@code false} otherwise.
	*/
	public boolean isConnected() {
	return mState == State.CONNECTED;
	}
	/**
	* Indicates whether network connectivity is possible. A network is unavailable
	* when a persistent or semi-persistent condition prevents the possibility
	* of connecting to that network. Examples include
	*
	* - The device is out of the coverage area for any network of this type.
	* - The device is on a network other than the home network (i.e., roaming), and
	* data roaming has been disabled.
	* - The device's radio is turned off, e.g., because airplane mode is enabled.
	*
	* @return {@code true} if the network is available, {@code false} otherwise
	*/
	public boolean isAvailable() {
	return mIsAvailable;
	}
	
	/**
	* Sets if the network is available, ie, if the connectivity is possible.
	* @param isAvailable the new availability value.
	*
	* {@hide}
	*/
	public void setIsAvailable(boolean isAvailable) {
	mIsAvailable = isAvailable;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	}
	
	public String getTypeName() {
	switch (mNetworkType) {
	case ConnectivityManager.TYPE_WIFI:
	return "WIFI";
	case ConnectivityManager.TYPE_MOBILE:
	return "MOBILE";
	default:
	return "<invalid>";
	}
	}
	
	SAMSUNG_PRIORART0005487, NetworkStateTracker
	/**
	* Turn the wireless radio off for a network.
	* @param turnOn {@code true} to turn the radio on, {@code false}
	*/
	public abstract boolean setRadio(boolean turnOn);
	
	/**
	* Returns an indication of whether this network is available for
	* connections. A value of {@code false} means that some quasi-permanent
	* condition prevents connectivity to this network.
	*/
	public abstract boolean isAvailable();
	
	SAMSUNG_PRIORART0005487, ConnectivityService
	/*

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	// The WifiStateTracker should appear first in the list
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;
	
	mActiveNetwork = null;
	mNumDnsEntries = 0;
	mTestMode = SystemProperties.get("cm.test.mode").equals("true")
	&& SystemProperties.get("ro.build.type").equals("eng");
	for (NetworkStateTracker t : mNetTrackers)
	t.startMonitoring();
	// Constructing this starts it too
	mWifiWatchdogService = new WifiWatchdogService(context, mWifiStateTracker);
	}
	
	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing
	* a connection to the preferred network is taken care of when we handle
	* the disconnect event from the non-preferred network

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* (see {@link #handleDisconnect(NetworkInfo)}). */ /** * Ensure that a network route exists to deliver traffic to the specified * host via the specified network interface. * @param networkType the type of the network over which traffic to the specified * host is to be routed * @param hostAddress the IP address of the host to which the route is desired * @return {@code true} on success, {@code false} on failure */ **Android 1.6** **SAMSUNG_PRIORART0005350, ConnectivityManager** /** * Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling * {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.) * - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network * is lost

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	@SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
	public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED =
	"android.net.conn.BACKGROUND_DATA_SETTING_CHANGED";
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	public void setNetworkPreference(int preference) {
	try {
	mService.setNetworkPreference(preference);
	} catch (RemoteException e) {
	}
	}
	
	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {
	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	return false;
	}
	}
	
	/**
	* Returns the value of the setting for background data usage. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005350, NetworkInfo
	
	/**
	* Indicates whether network connectivity is possible:
	*/
	private boolean mIsAvailable;
	
	/**
	* Return a human-readable name describe the type of the network,
	* for example "WIFI" or "MOBILE".
	* @return the name of the network type
	*/
	public String getTypeName() {
	return mTypeName;
	}
	
	/**
	* Indicates whether network connectivity exists or is in the process
	* of being established. This is good for applications that need to

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* do anything related to the network other than read or write data.
	* For the latter, call {@link #isConnected()} instead, which guarantees
	* that the network is fully usable.
	* @return {@code true} if network connectivity exists or is in the process
	* of being established, {@code false} otherwise.
	*/
	public boolean isConnectedOrConnecting() {
	return mState == State.CONNECTED || mState == State.CONNECTING;
	}
	
	/**
	* Indicates whether network connectivity is possible. A network is unavailable
	* when a persistent or semi-persistent condition prevents the possibility
	* of connecting to that network. Examples include
	*
	* - The device is out of the coverage area for any network of this type.
	* - The device is on a network other than the home network (i.e., roaming), and
	* data roaming has been disabled.
	* - The device's radio is turned off, e.g., because airplane mode is enabled.
	*
	* @return {@code true} if the network is available, {@code false} otherwise
	*/
	public boolean isAvailable() {
	return mIsAvailable;
	}
	
	/**
	* Indicates whether the device is currently roaming on this network.
	* When {@code true}, it suggests that use of data on this network
	* may incur extra costs.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* @return {@code true} if roaming is in effect, {@code false} otherwise.
	*/
	public boolean isRoaming() {
	return mIsRoaming;
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;
	
	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing
	* a connection to the preferred network is taken care of when we handle
	* the disconnect event from the non-preferred network

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

```
     * (see {@link #handleDisconnect(NetworkInfo)}).
     */
    private void enforcePreference() {
        if (mActiveNetwork == null)
            return;

        for (NetworkStateTracker t : mNetTrackers) {
            if (t == mActiveNetwork) {
                int netType = t.getNetworkInfo().getType();
                int otherNetType = ((netType == ConnectivityManager.TYPE_WIFI) ?
                        ConnectivityManager.TYPE_MOBILE :
                        ConnectivityManager.TYPE_WIFI);

                if (t.getNetworkInfo().getType() != mNetworkPreference) {
                    NetworkStateTracker otherTracker = mNetTrackers[otherNetType];
                    if (otherTracker.isAvailable()) {
                        teardown(t);
                    }
                }
            }
        }
    }

/**
     * @see ConnectivityManager#getBackgroundDataSetting()
     */
    public boolean getBackgroundDataSetting() {
        return Settings.Secure.getInt(mContext.getContentResolver(),
                Settings.Secure.BACKGROUND_DATA, 1) == 1;
    }
```

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {
	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}
	
	/**
	* See if the other network is available to fail over to.
	* If is not available, we enable it anyway, so that it
	* will be able to connect when it does become available,
	* but we report a total loss of connectivity rather than
	* report that we are attempting to fail over.
	*/
	NetworkInfo switchTo = null;
	if (newNet.isAvailable()) {
	mActiveNetwork = newNet;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	switchTo = newNet.getNetworkInfo();
	switchTo.setFailover(true);
	if (!switchTo.isConnectedOrConnecting()) {
	newNet.reconnect();
	}
	} else {
	newNet.reconnect();
	}
	
	if (info.getType() == ConnectivityManager.TYPE_MOBILE) {
	otherNet = mWifiStateTracker;
	} else /* info().getType() == TYPE_WIFI */ {
	otherNet = mMobileDataStateTracker;
	}
	
	int incrValue = ConnectivityManager.TYPE_MOBILE - ConnectivityManager.TYPE_WIFI;
	int stopValue = ConnectivityManager.TYPE_MOBILE + incrValue;
	
	SAMSUNG_PRIORART0005350, BatteryStats
	
	/**
	* A constant indicating a a wifi turn on timer
	*
	* {@hide}
	*/
	public static final int WIFI_TURNED_ON = 4;
	
	/**
	* A constant indicating a full wifi lock timer

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* * {@hide} */ public static final int FULL_WIFI_LOCK = 5; /** * A constant indicating a scan wifi lock timer * * {@hide} */ public static final int SCAN_WIFI_LOCK = 6; /** * A constant indicating a wifi multicast timer * * {@hide} */ public static final int WIFI_MULTICAST_ENABLED = 7; /** * A constant indicating an audio turn on timer * * {@hide} */ public static final int AUDIO_TURNED_ON = 7; /** * A constant indicating a video turn on timer * * {@hide}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	*/ public static final int VIDEO_TURNED_ON = 8; /** * Include all of the data in the stats, including previously saved data. */ public static final int STATS_TOTAL = 0; /** * Include only the last run in the stats. */ public static final int STATS_LAST = 1; /** * Include only the current run in the stats. */ public static final int STATS_CURRENT = 2; /** * Include only the run since the last time the device was unplugged in the stats. */ public static final int STATS_UNPLUGGED = 3; public abstract void noteWifiTurnedOnLocked(); public abstract void noteWifiTurnedOffLocked(); public abstract void noteFullWifiLockAcquiredLocked(); public abstract void noteFullWifiLockReleasedLocked(); public abstract void noteScanWifiLockAcquiredLocked(); public abstract void noteScanWifiLockReleasedLocked();

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	public abstract void noteWifiMulticastEnabledLocked(); public abstract void noteWifiMulticastDisabledLocked(); /** * Returns the time in microseconds that the screen has been on while the device was * running on battery. * * {@hide} */ public abstract long getScreenOnTime(long batteryRealtime, int which); public static final int SCREEN_BRIGHTNESS_DARK = 0; public static final int SCREEN_BRIGHTNESS_DIM = 1; public static final int SCREEN_BRIGHTNESS_MEDIUM = 2; public static final int SCREEN_BRIGHTNESS_LIGHT = 3; public static final int SCREEN_BRIGHTNESS_BRIGHT = 4; public static final int DATA_CONNECTION_NONE = 0; public static final int DATA_CONNECTION_GPRS = 1; public static final int DATA_CONNECTION_EDGE = 2; public static final int DATA_CONNECTION_UMTS = 3; public static final int DATA_CONNECTION_OTHER = 4; /** * Returns the time in microseconds that wifi has been on while the device was * running on battery. * * {@hide} */ public abstract long getWifiOnTime(long batteryRealtime, int which);

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	/** * Returns the time in microseconds that bluetooth has been on while the device was * running on battery. * * {@hide} */ public abstract long getBluetoothOnTime(long batteryRealtime, int which); /** * Return whether we are currently running on battery. */ public abstract boolean getIsOnBattery(); /** * Returns the time that the radio was on for data transfers. * @return the uptime in microseconds while unplugged */ public abstract long getRadioDataUptime(); /** * Returns the current battery realtime in microseconds. * * @param curTime the amount of elapsed realtime in microseconds. */ public abstract long getBatteryRealtime(long curTime); /** * Returns the battery percentage level at the last time the device was unplugged from power, or

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* the last time it booted on battery power. */ public abstract int getDischargeStartLevel(); **Android 2.2** **GOOG-HEADWATER-00000029, SAMSUNG_PRIORART0005353, ConnectivityManager** /** * Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling * {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.) * - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network * is lost * - Provide an API that allows applications to query the coarse-grained or fine-grained * state of the available networks * */ * A change in network connectivity has occurred. A connection has either * been established or lost. The NetworkInfo for the affected network is

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* sent as an extra; it should be consulted to see what kind of * connectivity event occurred. /** * Broadcast Action: The setting for **background data usage** has changed * values. Use {@link #getBackgroundDataSetting()} to get the current value. * <p> * If an application uses the network in the background, it should listen * for this broadcast and stop using the background data if the value is * false. */ @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION) public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED = "android.net.conn.BACKGROUND_DATA_SETTING_CHANGED"; /** * The Default Mobile data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_MOBILE = 0; /** * The Default WIFI data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_WIFI = 1;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* Returns the value of the setting **for background data usage**. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	/**
	* Sets the persisted value for enabling/disabling Mobile data.
	*
	* @param enabled Whether the mobile data connection should be
	* used or not.
	* @hide
	*/
	public void setMobileDataEnabled(boolean enabled) {
	try {
	mService.setMobileDataEnabled(enabled);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005353, NetworkStateTracker
	
	/**
	* Record the roaming status of the device, and if it is a change from the previous
	* status, send a notification to any listeners.
	* @param isRoaming {@code true} if the device is now roaming, {@code false}

'578 Claims	Android Device with One or More Apps

	* if it is no longer roaming.
	*/
	protected void setRoamingStatus(boolean isRoaming) {
	if (isRoaming != mNetworkInfo.isRoaming()) {
	mNetworkInfo.setRoaming(isRoaming);
	Message msg = mTarget.obtainMessage(EVENT_ROAMING_CHANGED, mNetworkInfo);
	msg.sendToTarget();
	}
	}
	
	public static final int EVENT_ROAMING_CHANGED = 5;
	
	SAMSUNG_PRIORART0005353, ThrottleManager
	/**
	* Class that handles throttling. It provides read/write numbers per interface
	* and methods to apply throttled rates.
	* {@hide}
	*/
	
	/**
	* returns a long of the byte count either read or written on the named interface
	* for the period described. Direction is either DIRECTION_RX or DIRECTION_TX and
	* period may only be PERIOD_CYCLE for the current cycle (other periods may be supported
	* in the future). Ago indicates the number of periods in the past to lookup - 0 means
	* the current period, 1 is the last one, 2 was two periods ago..
	* {@hide}
	*/
	public long getByteCount(String iface, int direction, int period, int ago) {
	try {
	return mService.getByteCount(iface, direction, period, ago);

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	} catch (RemoteException e) {
	return -1;
	}
	}
	
	/**
	* returns the number of bytes read+written after which a particular cliff
	* takes effect on the named iface. Currently only cliff #1 is supported (1 step)
	* {@hide}
	*/
	public long getCliffThreshold(String iface, int cliff) {
	try {
	return mService.getCliffThreshold(iface, cliff);
	} catch (RemoteException e) {
	return -1;
	}
	}
	SAMSUNG_PRIORART0005353, ConnectivityManagerMobileTest
	
	// help function to verify 3G connection
	public void verifyCellularConnection() {
	NetworkInfo extraNetInfo = cmActivity.mNetworkInfo;
	assertEquals("network type is not MOBILE", ConnectivityManager.TYPE_MOBILE,
	extraNetInfo.getType());
	assertTrue("not connected to cellular network", extraNetInfo.isConnected());
	assertTrue("no data connection", cmActivity.mState.equals(State.CONNECTED));
	}
	
	// Wait for the Wifi state to be DISABLED

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**

	waitForWifiState(WifiManager.WIFI_STATE_DISABLED, STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_WIFI, State.DISCONNECTED, STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_MOBILE, State.CONNECTED, STATE_TRANSITION_LONG_TIMEOUT);
	
	//Prepare for connectivity state verification
	NetworkInfo networkInfo = cmActivity.mCM.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
	cmActivity.setStateTransitionCriteria(ConnectivityManager.TYPE_MOBILE, networkInfo.getState(), NetworkState.DO_NOTHING, State.DISCONNECTED);
	networkInfo = cmActivity.mCM.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
	cmActivity.setStateTransitionCriteria(ConnectivityManager.TYPE_WIFI, networkInfo.getState(), NetworkState.TO_CONNECTION, State.CONNECTED);
	
	// Wait for Wifi to be connected and mobile to be disconnected
	waitForNetworkState(ConnectivityManager.TYPE_WIFI, State.CONNECTED, STATE_TRANSITION_LONG_TIMEOUT);
	waitForNetworkState(ConnectivityManager.TYPE_MOBILE, State.DISCONNECTED, STATE_TRANSITION_LONG_TIMEOUT);
	
	// Test case 5: test connectivity from 3G to airplane mode, then to 3G again
	
	// Test case 6: test connectivity with airplane mode Wifi connected

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	SAMSUNG_PRIORART0005353, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	boolean noMobileData = !getMobileDataEnabled();
	for (int netType : mPriorityList) {
	switch (mNetAttributes[netType].mRadio) {
	case ConnectivityManager.TYPE_WIFI:
	if (DBG) Slog.v(TAG, "Starting Wifi Service.");
	WifiStateTracker wst = new WifiStateTracker(context, mHandler);
	WifiService wifiService = new WifiService(context, wst);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);
	wifiService.startWifi();
	mNetTrackers[ConnectivityManager.TYPE_WIFI] = wst;
	wst.startMonitoring();
	
	break;
	case ConnectivityManager.TYPE_MOBILE:
	mNetTrackers[netType] = new MobileDataStateTracker(context, mHandler,
	netType, mNetAttributes[netType].mName);
	mNetTrackers[netType].startMonitoring();
	if (noMobileData) {
	if (DBG) Slog.d(TAG, "tearing down Mobile networks due to setting");
	mNetTrackers[netType].teardown();
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	break;
	default:
	Slog.e(TAG, "Trying to create a DataStateTracker for an unknown radio type " +
	mNetAttributes[netType].mRadio);
	continue;
	}
	}
	
	/**
	* Sets the preferred network.
	* @param preference the new preference
	*/
	public synchronized void setNetworkPreference(int preference) {
	enforceChangePermission();
	if (ConnectivityManager.isNetworkTypeValid(preference) &&
	mNetAttributes[preference] != null &&
	mNetAttributes[preference].isDefault()) {
	if (mNetworkPreference != preference) {
	persistNetworkPreference(preference);
	mNetworkPreference = preference;
	enforcePreference();
	}
	}
	}
	
	/**
	* Return NetworkInfo for the active (i.e., connected) network interface.
	* It is assumed that at most one network is active at a time. If more
	* than one is active, it is indeterminate which will be returned.
	* @return the info for the active network, or {@code null} if none is

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* active
	*/
	public NetworkInfo getActiveNetworkInfo() {
	enforceAccessPermission();
	for (int type=0; type <= ConnectivityManager.MAX_NETWORK_TYPE; type++) {
	if (mNetAttributes[type] == null || !mNetAttributes[type].isDefault()) {
	continue;
	}
	NetworkStateTracker t = mNetTrackers[type];
	NetworkInfo info = t.getNetworkInfo();
	if (info.isConnected()) {
	if (DBG && type != mActiveDefaultNetwork) Slog.e(TAG,
	"connected default network is not " +
	"mActiveDefaultNetwork!");
	return info;
	}
	}
	return null;
	}
	
	// TODO - move this into the MobileDataStateTracker
	int usedNetworkType = networkType;
	if(networkType == ConnectivityManager.TYPE_MOBILE) {
	if (!getMobileDataEnabled()) {
	if (DBG) Slog.d(TAG, "requested special network with data disabled - rejected");
	return Phone.APN_TYPE_NOT_AVAILABLE;
	}
	if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_MMS)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_MMS;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_SUPL)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_SUPL;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_DUN)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_DUN;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_HIPRI)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_HIPRI;
	}
	}
	
	int usedNetworkType = networkType;
	if (networkType == ConnectivityManager.TYPE_MOBILE) {
	if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_MMS)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_MMS;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_SUPL)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_SUPL;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_DUN)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_DUN;
	} else if (TextUtils.equals(feature, Phone.FEATURE_ENABLE_HIPRI)) {
	usedNetworkType = ConnectivityManager.TYPE_MOBILE_HIPRI;
	}
	}
	
	/**
	* @see ConnectivityManager#getBackgroundDataSetting()
	*/
	public boolean getBackgroundDataSetting() {
	return Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, 1) == 1;
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {
	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA,
	allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}
	
	/**
	* @see ConnectivityManager#getMobileDataEnabled()
	*/
	public boolean getMobileDataEnabled() {
	enforceAccessPermission();
	boolean retVal = Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.MOBILE_DATA, 1) == 1;
	if (DBG) Slog.d(TAG, "getMobileDataEnabled returning " + retVal);
	return retVal;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	} **SAMSUNG_PRIORART0005353, BatteryStats** /** * A class providing access to battery usage statistics, including information on * wakelocks, processes, packages, and services. All times are represented in microseconds * except where indicated otherwise. * @hide */ /** * A constant indicating a a wifi turn on timer * * {@hide} */ public static final int WIFI_TURNED_ON = 4; /** * A constant indicating an audio turn on timer * * {@hide} */ public static final int AUDIO_TURNED_ON = 7; /** * A constant indicating a video turn on timer * * {@hide}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	*/
	public static final int VIDEO_TURNED_ON = 8;
	
	public static final int SIGNAL_STRENGTH_NONE_OR_UNKNOWN = 0;
	public static final int SIGNAL_STRENGTH_POOR = 1;
	public static final int SIGNAL_STRENGTH_MODERATE = 2;
	public static final int SIGNAL_STRENGTH_GOOD = 3;
	public static final int SIGNAL_STRENGTH_GREAT = 4;
	
	static final String[] SIGNAL_STRENGTH_NAMES = {
	"none", "poor", "moderate", "good", "great"
	};
	
	public static final int DATA_CONNECTION_NONE = 0;
	public static final int DATA_CONNECTION_GPRS = 1;
	public static final int DATA_CONNECTION_EDGE = 2;
	public static final int DATA_CONNECTION_UMTS = 3;
	public static final int DATA_CONNECTION_OTHER = 4;
	
	GOOG-HEADWATER-00000092, Google I/O 2009 - Coding for Life -- Battery Life, That Is (June 2, 2009)
	
	Google I/O, 2009

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

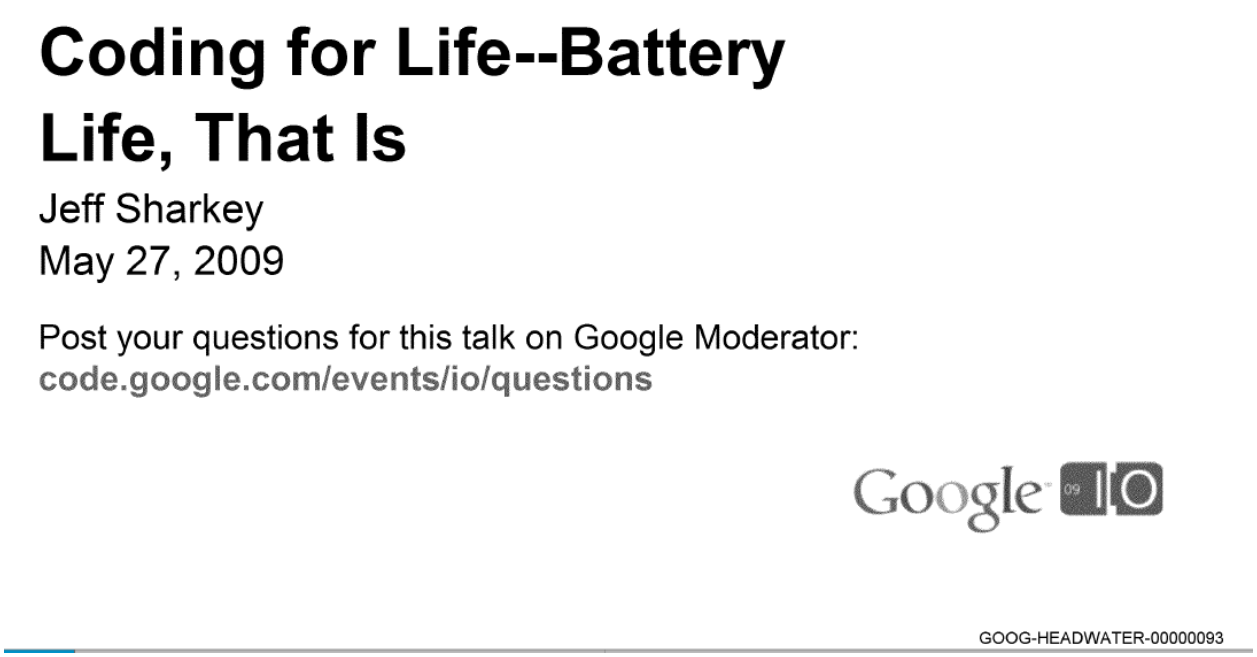
'578 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 2: 

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

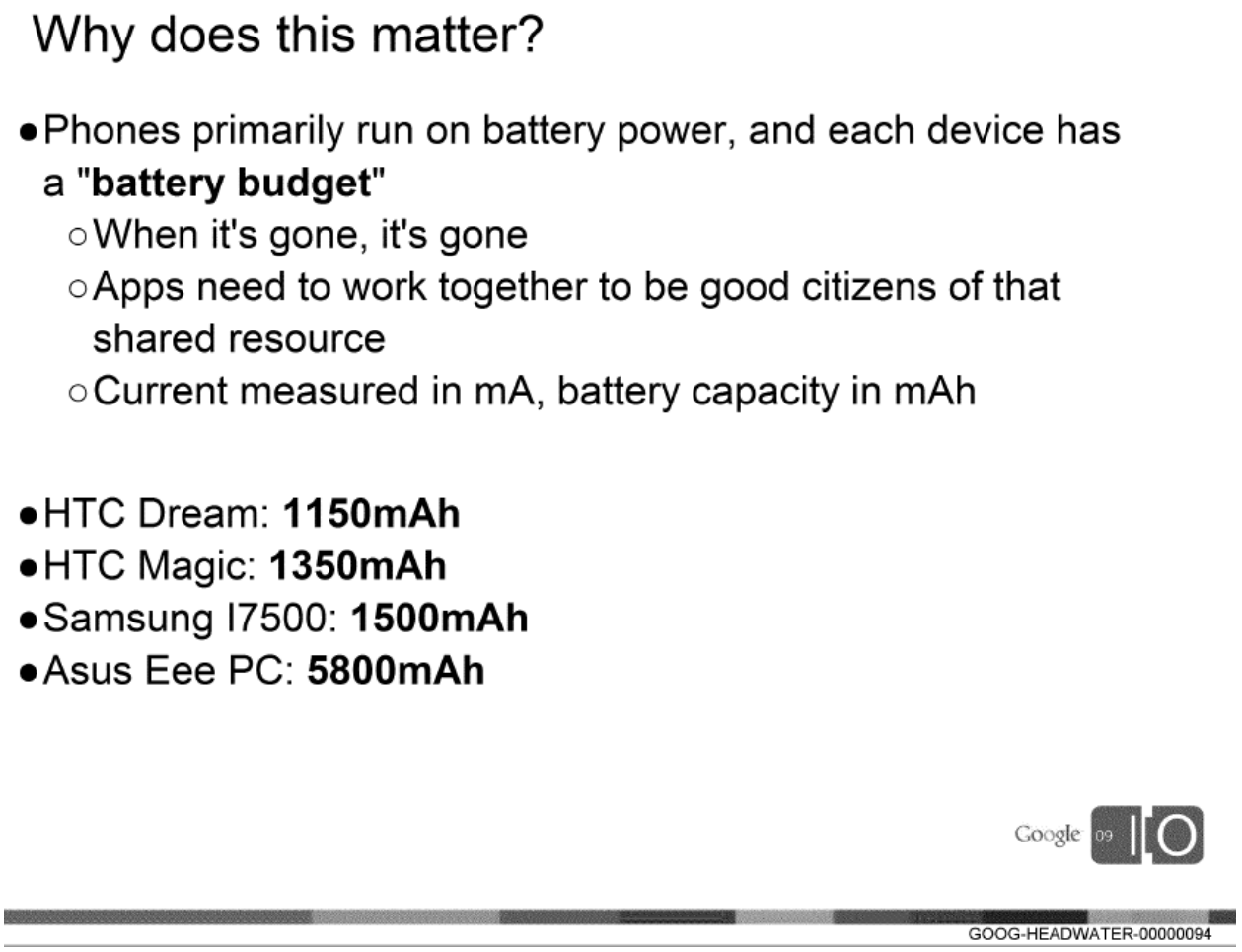
'578 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 3: 

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

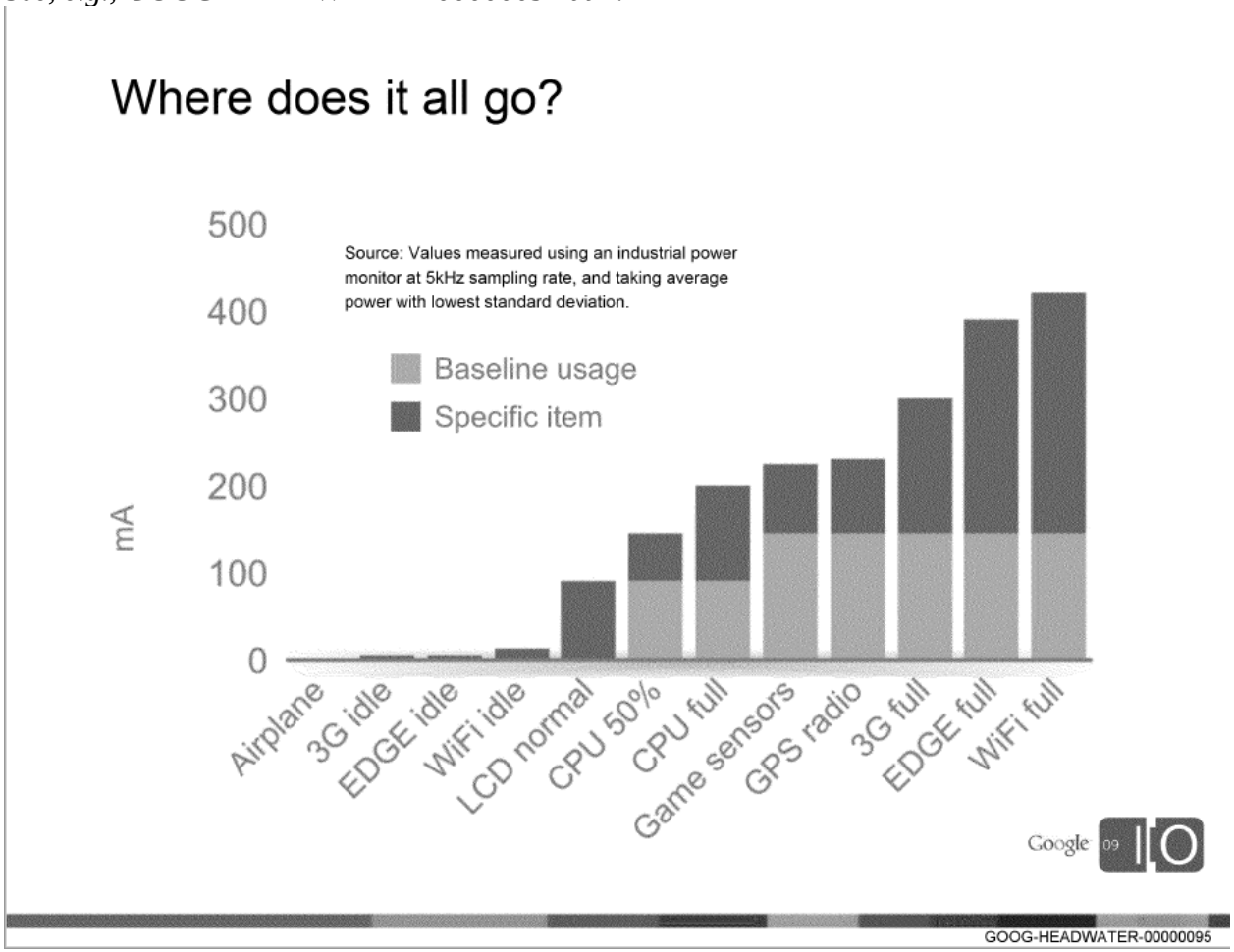
'578 Claims	**Android Device with One or More Apps**
	See, e.g., GOOG-HEADWATER-00000092 at 4: 

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

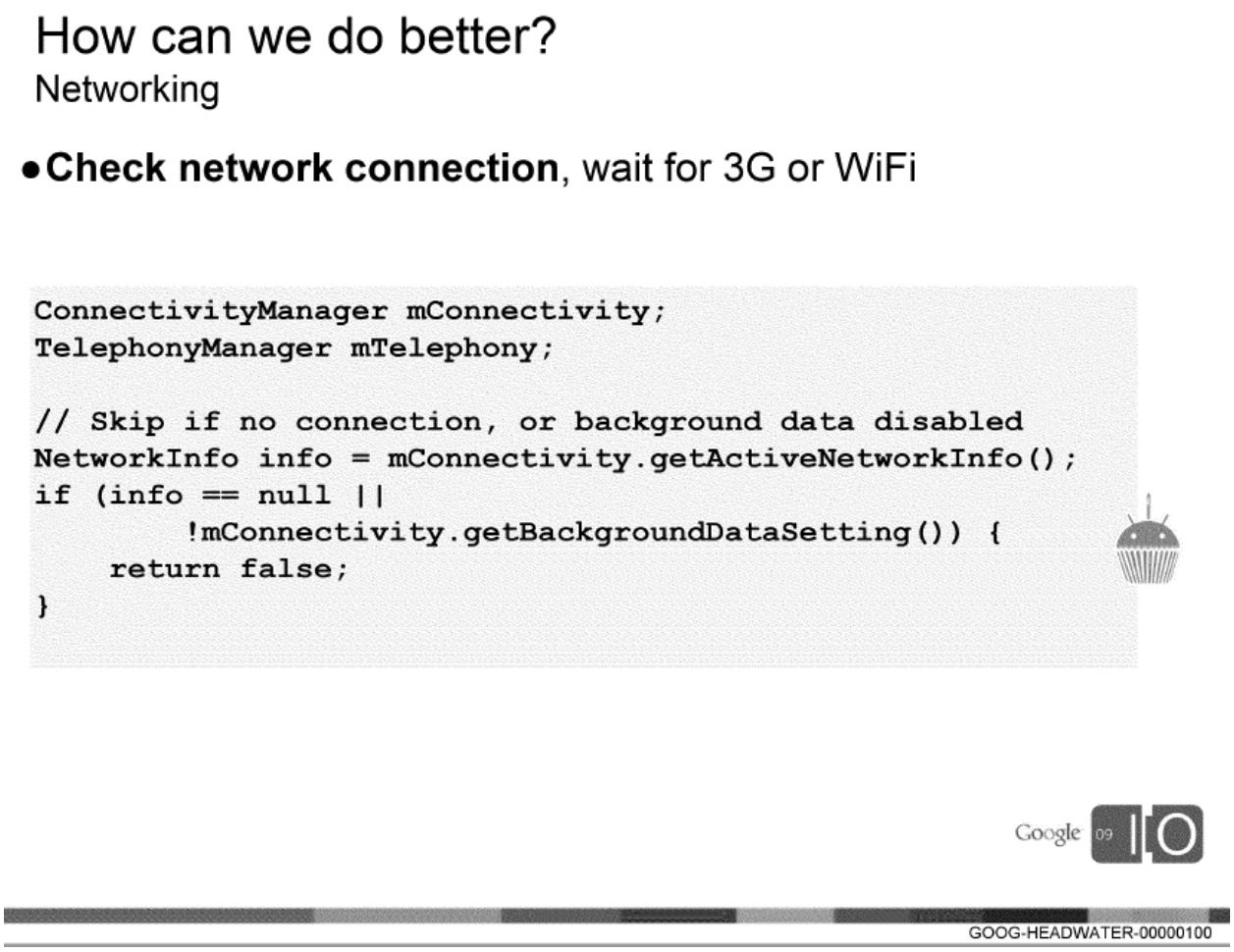
'578 Claims	Android Device with One or More Apps
	See, e.g., GOOG-HEADWATER-00000092 at 9:  *See, e.g.*, GOOG-HEADWATER-00000092 at 11:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

How can we do better?
Networking

- **Check network connection**, wait for 3G or WiFi

```
// Only update if WiFi or 3G is connected and not roaming
int netType = info.getType();
int netSubtype = info.getSubtype();

if (netType == ConnectivityManager.TYPE_WIFI) {
    return info.isConnected();
} else if (netType == ConnectivityManager.TYPE_MOBILE
        && netSubtype == TelephonyManager.NETWORK_TYPE_UMTS
        && !mTelephony.isNetworkRoaming()) {
    return info.isConnected();
} else {
    return false;
}
```

Google 09 I/O

GOOG-HEADWATER-00000101

See, e.g., GOOG-HEADWATER-00000092 at 16:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

How can we do better?

Foreground apps

- **Wakelocks are costly** if forgotten
  - Pick the lowest level possible, and use specific timeouts to work around unforseen bugs
  - Consider using android:keepScreenOn to ensure correctness

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:keepScreenOn="true">
```

Google 09 I/O

GOOG-HEADWATER-00000107

See, e.g., GOOG-HEADWATER-00000092 at 18:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

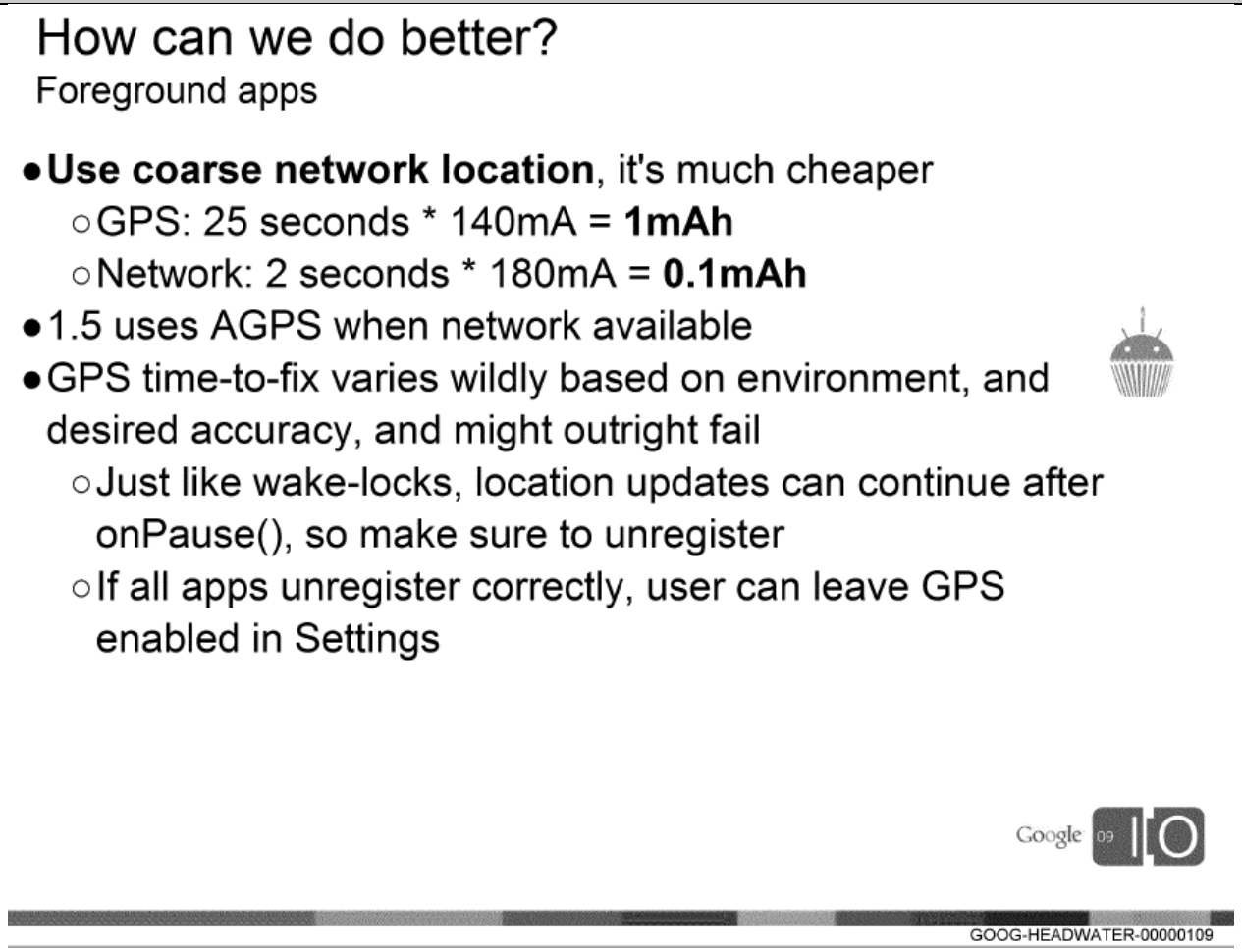
'578 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 20:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

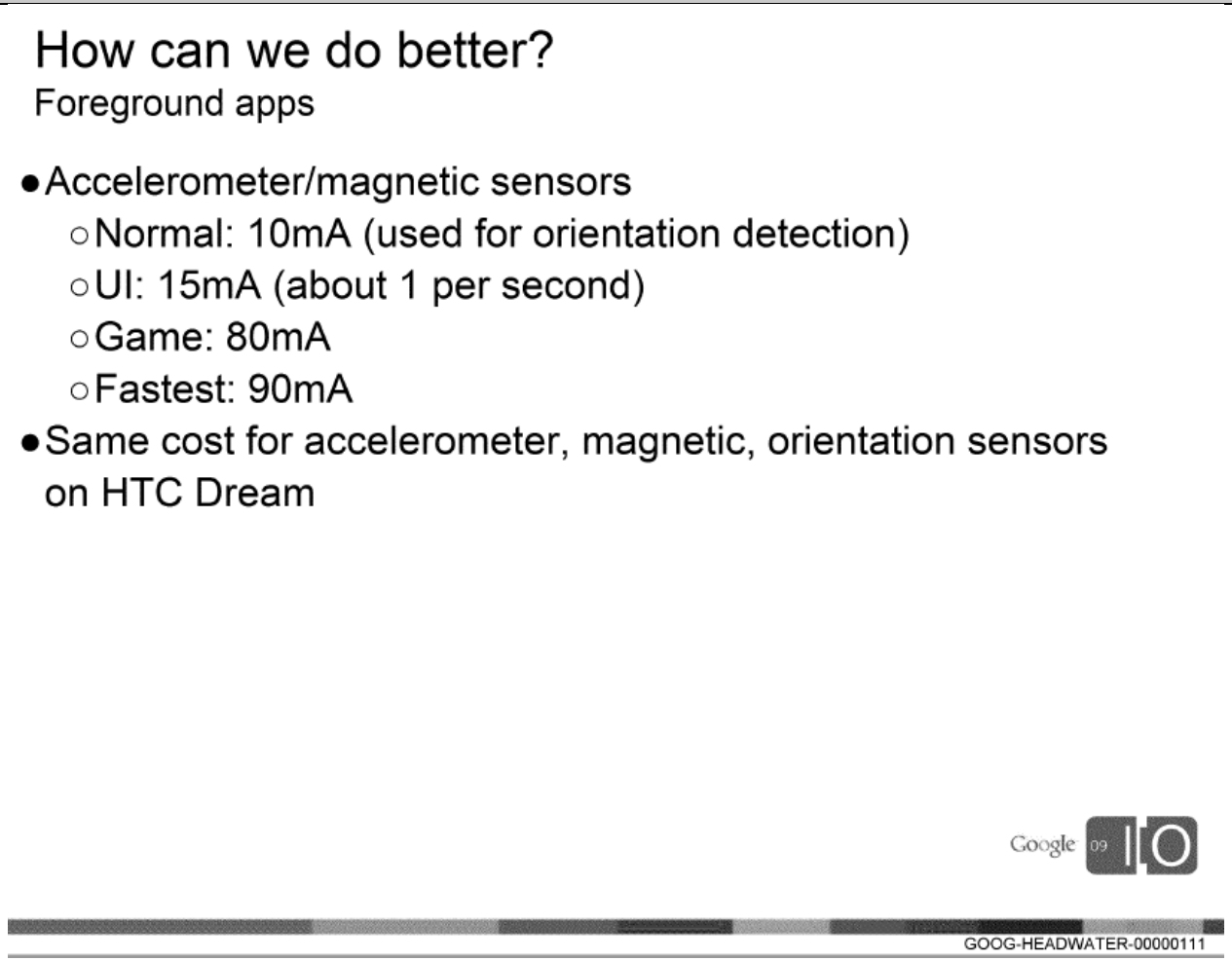
'578 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 22:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

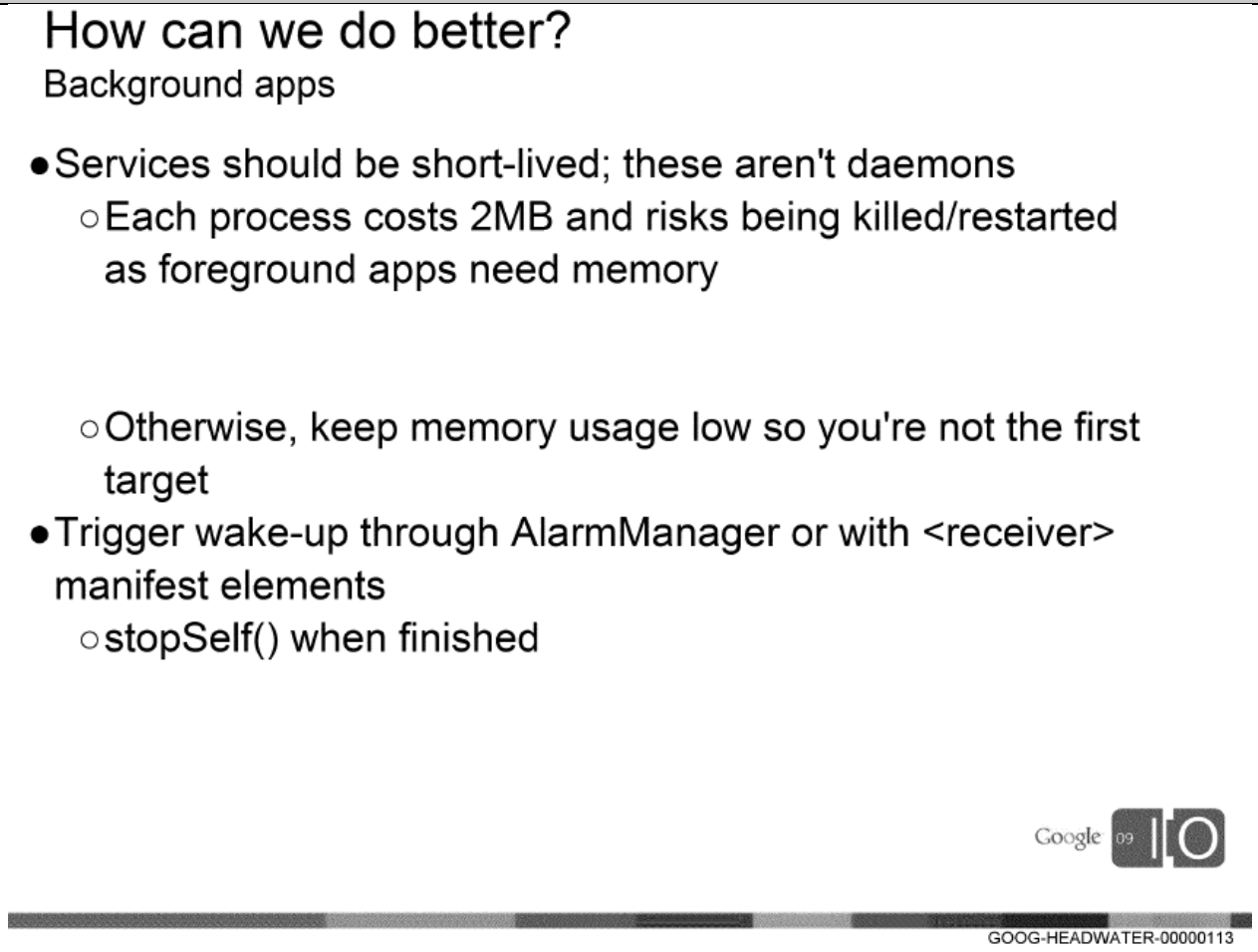
'578 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 26:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

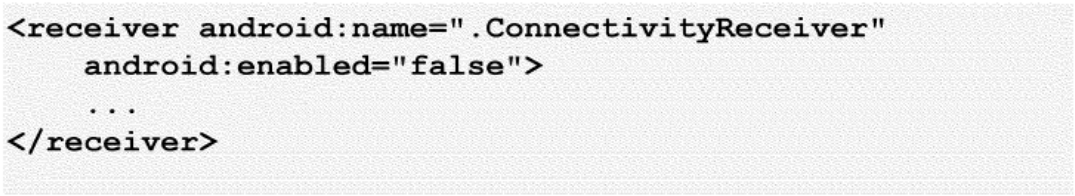
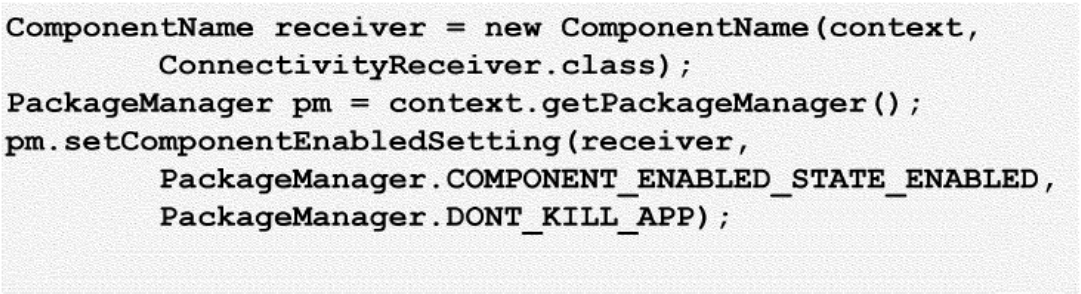
'578 Claims	Android Device with One or More Apps
	How can we do better? Background apps ● Dynamically enabling/disabling <receiver> components in manifest, especially when no-ops   Google 09 I/O GOOG-HEADWATER-00000117

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	See, e.g., GOOG-HEADWATER-00000092 at 27:
	How can we do better?
	Background apps
	● Checking current battery and network state before running a full update
	`public void onCreate() {`
	`  // Register for sticky broadcast and send default`
	`  registerReceiver(mReceiver, mFilter);`
	`  mHandler.sendEmptyMessageDelayed(MSG_BATT, 1000);`
	`}`
	`IntentFilter mFilter =`
	`  new IntentFilter(Intent.ACTION_BATTERY_CHANGED);`
	`BroadcastReceiver mReceiver = new BroadcastReceiver() {`
	`  public void onReceive(Context context, Intent intent) {`
	`  // Found sticky broadcast, so trigger update`
	`  unregisterReceiver(mReceiver);`
	`  mHandler.removeMessages(MSG_BATT);`
	`  mHandler.obtainMessage(MSG_BATT, intent).sendToTarget();`
	`  }`
	`};`
	Google 09 IO
	GOOG-HEADWATER-00000118
	See, e.g., GOOG-HEADWATER-00000092 at 29:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

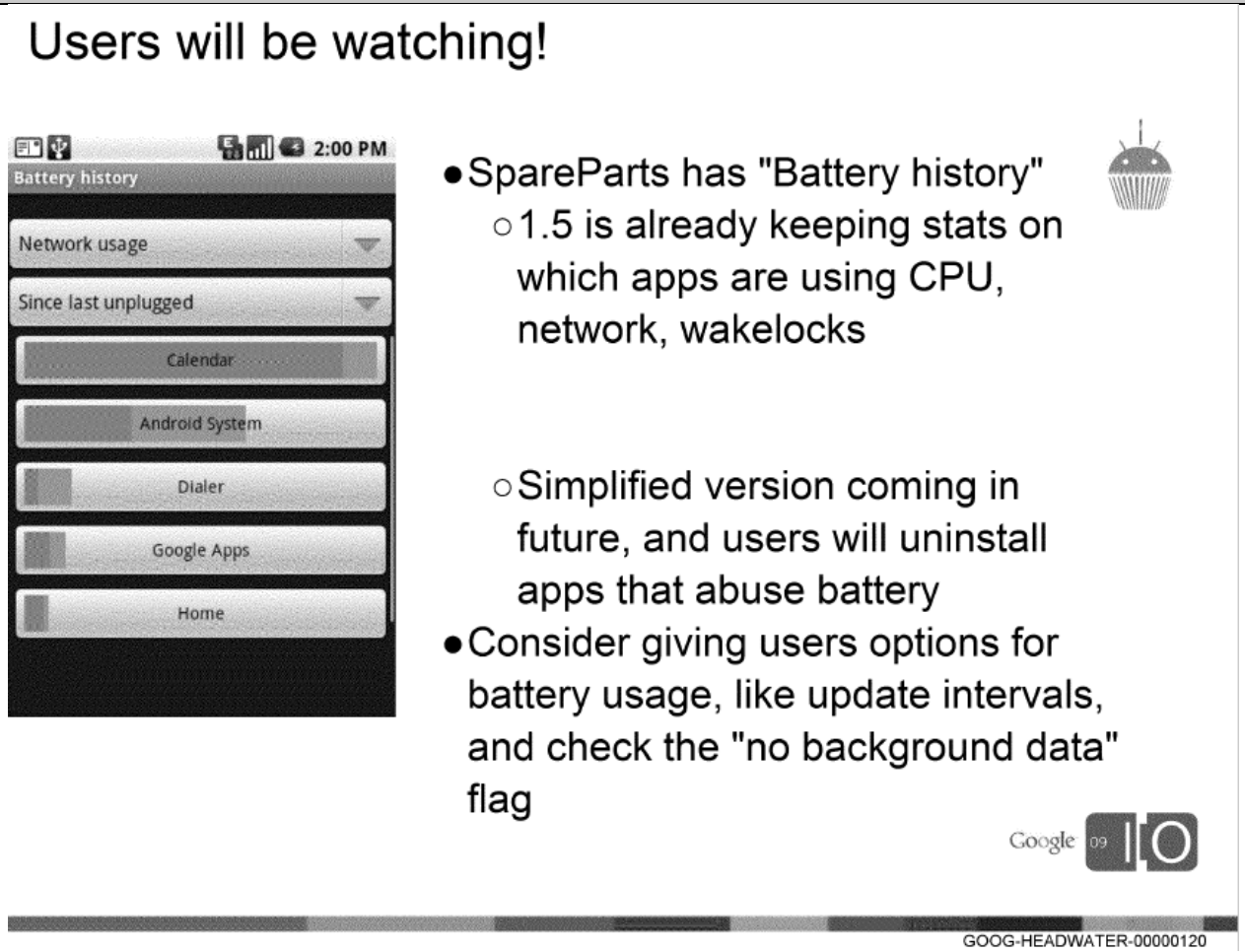
'578 Claims	Android Device with One or More Apps
	 *See, e.g.*, GOOG-HEADWATER-00000092 at 30:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

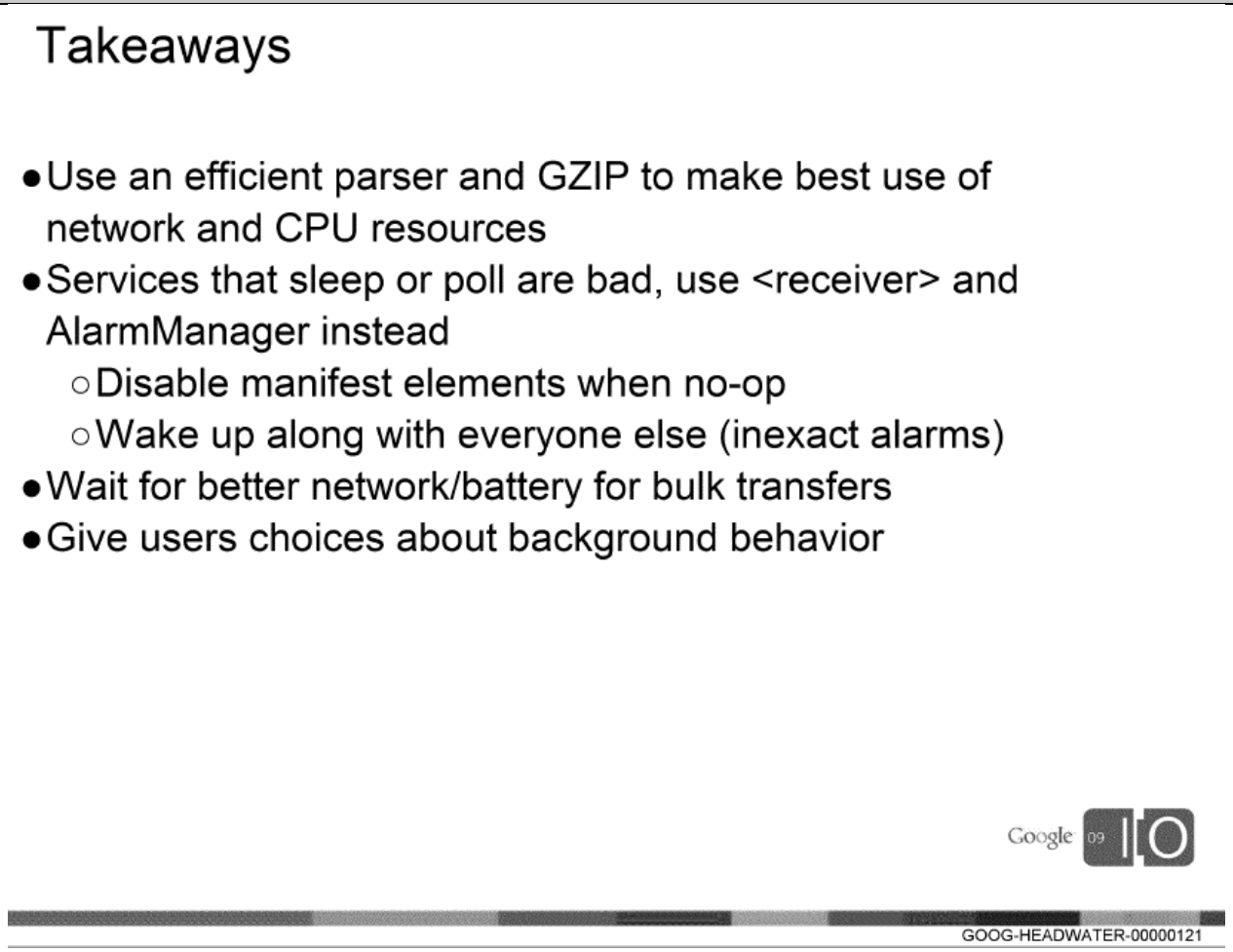
'578 Claims	Android Device with One or More Apps
	
[1b] a wireless local area network (WLAN)	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
modem to communicate data for Internet service activities between the device and at least one WLAN, when configured for and connected to the at least one WLAN;	*See, e.g.,* the disclosures identified for claims [1 pre] – [1a]. **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Manage Wifi If this setting is selected, then Green Power will regularly turn on and off the Wifi connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Wifi as it is, never turning it on or off. Please note that if you manually switches off the Wifi, then Green Power will unselect the "Manage Wifi" setting in order not to automatically switch on the Wifi again despite your manual action. Then, if you switch back on the Wifi or reselect "Manage Wifi" setting, Green Power will resume managing Wifi connection.").
[1c] a non-transitory memory to store a differential traffic control policy applicable to data communicated for Internet service activities using the WWAN modem and the at least one WWAN, but not applicable to data communicated for Internet service activities using the	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: **Nexus One** *See, e.g.,* SAMSUNG_PRIORART0000001 (Nexus) at 65-68:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
WLAN modem and the at least one WLAN;	**Connecting to networks and devices** Your phone can connect to a variety of networks and devices, including mobile networks for voice and data transmission, Wi-Fi data networks, and Bluetooth devices, such as headsets. You can also connect your phone to a computer, to transfer files from your phone's microSD card. **In this section** "Connecting to mobile networks" on page 66 "Connecting to Wi-Fi networks" on page 68 "Connecting to Bluetooth devices" on page 71 "Connecting to a computer via USB" on page 74 "Connecting to virtual private networks" on page 76 "Working with secure certificates" on page 78

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Connecting to mobile networks When you assemble your phone with a SIM card from your wireless service provider (see "Installing the battery, SIM, and microSD card" on page 18), your phone is configured to use your provider's mobile networks for voice calls and for transmitting data. Your phone is configured to work with many mobile carriers' networks. If your phone does not connect to a network when you insert a SIM card and turn it on, contact your carrier to obtain the details of its access point name. See "To edit or create a new access point" on page 71. Different locations may have different mobile networks available. Initially, your phone is configured to use the fastest mobile network available for data. But you can configure your phone to use only a slower 2G network for data, to extend the life of your battery between charges. You can also configure your phone to access a different set of networks entirely, or to behave in special ways when roaming. The icons in the Status bar indicate which kind of data network you're connected to and the voice and data network signal strength. Connected to the fastest 3G networks (UMTS or HSDPA) Connected to the second-fastest network (EDGE) Connected to a 2G network (GPRS) The more bars are lit, the stronger the wireless signal Connected to another wireless service provider's network (roaming) When you're connected to slower networks, you may want to postpone using your phone for data-intensive tasks until you are connected to a faster network again, or find a Wi-Fi network to connect to. See "Connecting to Wi-Fi networks" on page 68. **To determine what network you're using** 1 Press **Home**, press **Menu**, and touch **Settings** to open the Settings application. 2 Touch **Wireless & networks** › **Mobile networks** › **Access Point Names**. The name of the wireless service provider you're currently registered with is selected in the list.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	To disable data when roaming You can prevent your phone from transmitting data over other carriers' mobile networks when you leave an area that is covered by your carrier's networks. This is useful for controlling expenses if your cell plan doesn't include data roaming. 1 Press **Home**, press **Menu**, and touch **Settings**, to open the Settings application. 2 Touch **Wireless & networks** > **Mobile networks** and uncheck **Data roaming**. With **Data roaming** unchecked, you can still transmit data with a Wi-Fi connection.See "Connecting to Wi-Fi networks" on page 68. **To limit your data connection to 2G networks** You can extend your battery life by limiting your data connections to 2G networks (GPRS or EDGE). When you are connected to a 2G network, you may want to postpone activities that transmit a lot of data, such as sending, uploading, or downloading pictures or video, until you are connected to a faster mobile or other wireless network. 1 Press **Home**, press **Menu**, and touch **Settings** to open the Settings application. 2 Touch **Wireless & networks** > **Mobile networks** and check **Use only 2G networks**. **To edit or create a new access point** If you and your wireless service provider determine that you need to change the settings of your current access point name (APN) or to create a new one, you must obtain the APN and detailed settings from your provider. 1 Press **Home**, press **Menu**, and touch **Settings** to open the Settings application. 2 Touch **Wireless & networks** > **Mobile networks** > **Access Point Names**. 3 Touch an existing APN to edit it. Or press **Menu** and touch **New APN**. Enter the APN settings that you obtained from your wireless service provider by touching each setting that you need to edit. 4 When you're finished, press **Menu** and touch **Save**. 5 If you created a new APN, touch it in the APNs screen to start using it.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Connecting to Wi-Fi networks Wi-Fi is a wireless networking technology that can provide Internet access at distances of up to 100 meters, depending on the Wi-Fi router and your surroundings. To use Wi-Fi on your phone, you access a wireless access point, or "hotspot." Some access points are open and you can simply connect to them. Others are hidden or implement other security features, so you must configure your phone so it can connect to them. There are numerous systems for securing Wi-Fi connections, including some that rely on secure certificates or other schemes to ensure that only authorized users can connect. For information about installing secure certificates, see "Working with secure certificates" on page 78. Turn off Wi-Fi when you're not using it, to extend the life of your battery. The Status bar displays icons that indicate Wi-Fi status. Connected to a Wi-Fi network (waves indicate connection strength) Notification that an open Wi-Fi network is in range When you connect to a Wi-Fi network, the phone obtains a network address and other information it needs from the network, using the DHCP protocol. To configure the phone with a fixed IP address and other advanced settings, press **Menu** and touch **Advanced.** See "Advanced Wi-Fi settings screen" on page 310. **To turn Wi-Fi on and connect to a Wi-Fi network** If you're adding a Wi-Fi network when first setting up your phone, Wi-Fi is turned on automatically, so you can skip to step 4. 1 Press **Home**, press **Menu**, and touch **Settings**. 2 Touch **Wireless & networks** > **Wi-Fi settings**. **JuiceDefender App**

'578 Claims	Android Device with One or More Apps
	SAMSUNG_PRIORART0000335 (Ruddock) ("Juice Defender is a battery conservation app. It uses various triggers, rules, and timers to control how often your device utilizes 3G/EDGE APN's (data connections) as well as WiFi. These data connections are the number one drainers of battery life when your phone is idle, so Juice Defender allows you to decide when, where, and how often you want them to be active."). **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Manage Wifi If this setting is selected, then Green Power will regularly turn on and off the Wifi connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Wifi as it is, never turning it on or off. Please note that if you manually switches off the Wifi, then Green Power will unselect the "Manage Wifi" setting in order not to automatically switch on the Wifi again despite your manual action. Then, if you switch back on the Wifi or reselect "Manage Wifi" setting, Green Power will resume managing Wifi connection. Manage Mobile Network If this setting is selected, then Green Power will regularly turn on and off the Mobile Network connection, based on the durations specified in the settings below. If this setting is not selected, then Green Power will leave the Mobile Network as it is, never turning it on or off. Please note that in order for Green Power to turn on / off Mobile Network, this one has to be manually enabled by the user first in the phone settings (Wireless & networks → Mobile Network) or in Green

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
	Power settings (Global wireless settings → Mobile Network) . Green Power can't itself turn on Mobile Network as this is a limitation of the Android system for security and cost reasons."). In addition, *see, e.g.*, the disclosures identified for claims [1 pre] – [1a].
[1d] a user interface to allow a user to set one or more of a plurality of aspects of the differential traffic control policy to select one or more applications that are only allowed to utilize the at least one WWAN for Internet service activities when those applications are classified as interacting with a user in the device user interface foreground; and	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1c]. **Nexus One** *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 115-117:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Configuring account sync and display options You can configure background data use and synchronization options for all of the applications on your phone. You can also configure what kinds of data you synchronize for each account. Some applications, such as Gmail and Calendar, have their own synchronization settings. Some applications, such as Contacts and Gmail, can sync data from multiple applications. Others, such as Calendar, sync data only from the first Google Account you sign into on your phone, or from an account associated specifically with that application. For some accounts, syncing is two-directional; changes that you make to the information on your phone are made to the copy of that information on the web. Your Google Account works this way. Other accounts support only one-way sync; the information on your phone is read-only. You can use the Contacts display options to configure the kinds of contacts that are displayed, as described in "Changing which contacts are displayed" on page 106. **To configure general sync settings** 1 Open the Accounts & Sync Settings screen. You can do this in Contacts by pressing **Menu** and touching **Accounts**, or directly in the Settings application (press **Home**, press **Menu**, and touch **Settings**).

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

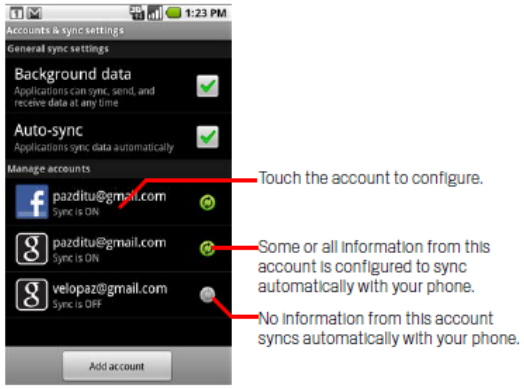
'578 Claims	Android Device with One or More Apps
	The screen displays your current sync settings and a list of your current accounts.  indicates that some or all of an account's information is configured to sync automatically with your phone. indicates that none of an account's information is configured to sync automatically with your phone. 2 Check or uncheck **Background data** to control whether applications and services can transmit data when you are not working with them directly (that is, when they are running in the background). If you uncheck this option, Gmail stops receiving new mail, Calendar stops syncing events, and so on, until you touch the **Refresh** menu item or send an email. 3 Check or uncheck **Auto-sync** to control whether changes you make to information on the phone or on the web are automatically synced with each other. For example, when this option is checked, changes that you make in Contacts on the phone are automatically made in Google Contacts on the web. If you uncheck this option, you may be able to use an application's tools to sync data manually. See "To sync information manually" on page 117.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

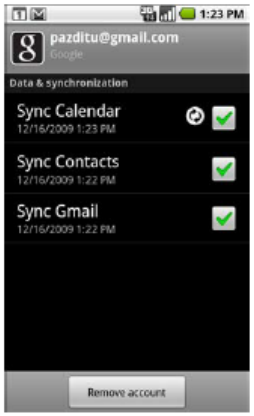
'578 Claims	Android Device with One or More Apps
	To sync information manually 1 Open the Accounts & Sync Settings screen. 2 Touch the account whose data you want to sync. 3 Press **Menu** and touch **Sync now**. **To change an account's sync settings** 1 Open the Accounts & Sync Settings screen. 2 Touch the account whose sync settings you want to change. The Data and Synchronization screen opens, displaying a list of the kinds of information the account can sync. Checked items are configured to sync to your phone.  3 Check or uncheck the kinds of information you want to sync to the phone. Unchecking an option does not remove the information from your phone; it simply stops it from syncing automatically. To remove the information previously synced for the account, you must remove the account. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 218:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Open in background Check to open new windows in the background when you touch & hold a link and touch **Open in new window**. This is useful when you are working with windows that take a long time to download and display. Press **Menu**, touch **Windows**, and then touch the new window to view it. Uncheck if you prefer new windows that you open in this way to open in place of the current window. See "To switch Browser windows" on page 213. **Set home page** Opens a dialog where you can enter the URL of a page that you want to open whenever you open a new Browser window. If you prefer to open new Browser windows more quickly, by not opening any page by default, leave the dialog blank. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 286: **Refresh settings** These settings control how frequently the information on the News & Weather widget is updated. In addition to using these settings, adding or removing the News & Weather widget from the Home screen also turns **Auto-refresh** on or off. For more about working with widgets, see "Customizing the Home screen" on page 58. **Auto-refresh** Check to have News & Weather update information automatically, at the frequency you set with **Refresh interval**. Uncheck to update the news and weather only when you press **Menu** and touch **Refresh**. For automatic refresh to work, you must also have Background Data turned on in the Settings application. See "Accounts & sync settings" on page 320. *See, e.g.*, SAMSUNG_PRIORART0000001 (Nexus) at 320:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

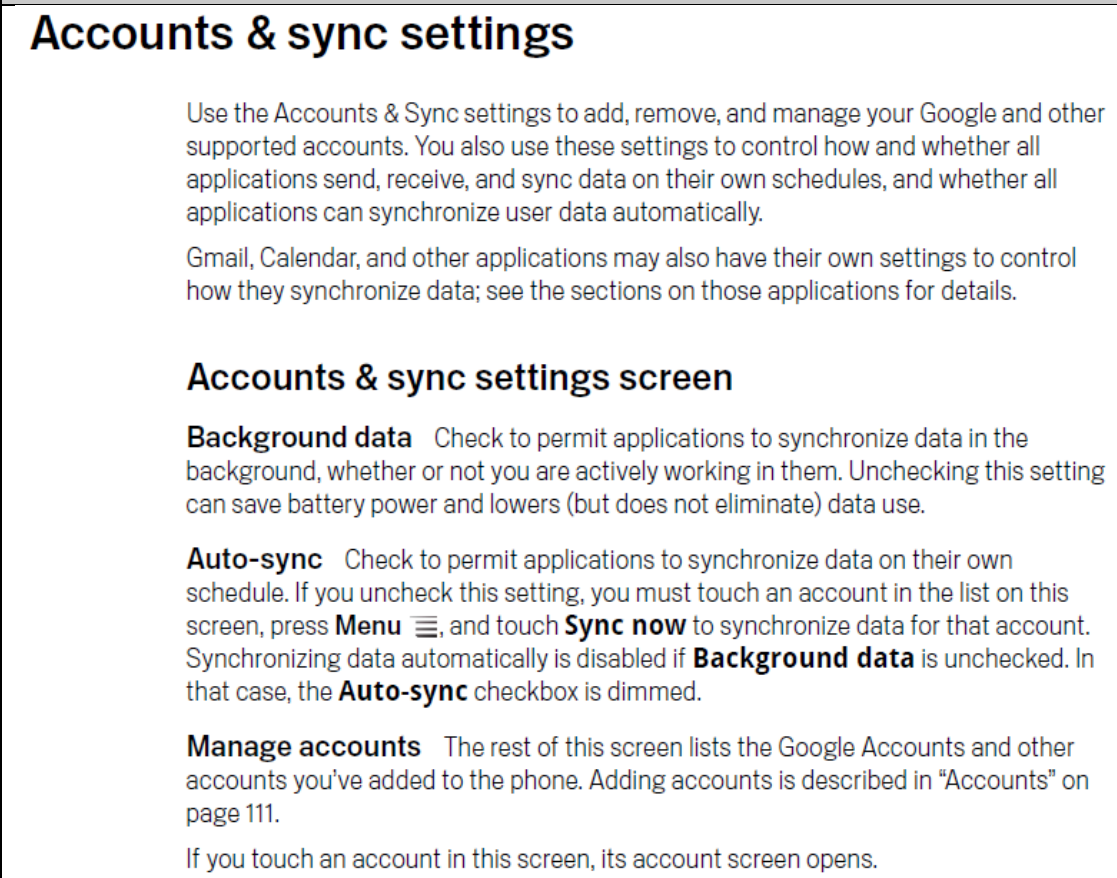
'578 Claims	Android Device with One or More Apps
	 **JuiceDefender App** SAMSUNG_PRIORART0000379 (Latedroid):

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	JuiceDefender **JuiceDefender** saves battery power (*lots of it!*) by controlling the device data connection and/or WiFi. You can schedule regular APN/WiFi activation to let background data sync occur and have APN/WiFi enabled while the screen is on. It also helps in minimizing distractions ;) The *Easy Mode* is a no-fuss one-click way to let your battery last longer - much longer. Just enable JuiceDefender by clicking on the big button and you're ready to go! If you want more fine-grained control, try *Advanced Mode*, where you can configure all JuiceDefender features to your liking.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	There are 5 *triggers* for the enable/disable behaviour: **Battery** - when battery level gets low (less than 15%), disable APN/WiFi, and re-enable them when battery level is restored. APN/WiFi will also be enabled while the device is being recharged. **Schedule** - regularly enable APN/WiFi for a short period of time, to *let background data sync* occur (email, Twitter, Facebook, stock quotes...). If *Quick* is disabled APN/WiFi stays enabled for a longer period, useful if your data connection is very slow or you need to sync lots of data. **Night schedule** (requires *UltimateJuice*) - disable APN/WiFi during night time; you can also optionally put the phone in Silent Mode. **Screen** - enable APN/WiFi *while the screen is on* to allow browsing, tweeting, procrastination and general internet-powered enjoyment, regardless of scheduled events and battery level. **Location** (requires *UltimateJuice*) - this trigger controlled by the *'AutoWiFi'* button. It disables WiFi when the device is not in range of any known WiFi network. The location is determined via the cellular network, so it's usually quite coarse. It's a fully automatic set-it-and-forget-it WiFi manager! The *priority order* of the triggers is 1) location (WiFi only), 2) screen, 3) battery, 4) night schedule, 5) schedule - this means, for example, that when the screen is on APN/WiFi will be enabled even when the battery is low, or that the regular schedule won't occur during the night period. SAMSUNG_PRIORART0000361 (Purdy):

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	 Android: Most phones don't make it easy to switch cellular data connection on and off, even if doing so really helps save your battery. JuiceDefender toggles wireless data and Wi-Fi on and off every so often to preserve power. The whole point of a smartphone with Google apps baked in is constant connectivity, of course, and you don't want to shut off access to your email, Google Voice messages, and other online services. But when you're walking around, at your office desk, and generally not actively using your phone, you probably don't need your phone to check in every minute with the mothership. JuiceDefender lets you set a time interval—5 minutes, 15, 30, an hour, two hours—at which its background process will re-enable your carrier APN, see if there are new messages or data coming in, and then shut off again. You can also set similar Wi-Fi connectivity rules, or only have web data enabled when you've got your screen on. Besides the battery savings, those who like to parse out their email checks and avoid minute-by-minute distractions see some benefit here, too. *See, e.g.*, SAMSUNG_PRIORART0000351 (Configuration-Translated): "• **Schedule:** used to configure how often Juice Defender will activate the APN / WIFI to synchronize emails, social networks, etc .., we select the interval in the bar of your submenu for example 5 minutes every half hour also has the **Quick** button to minimize the connection time for the previous example

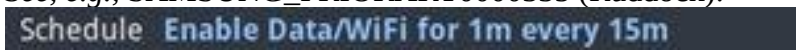
'578 Claims	Android Device with One or More Apps
	would be 2 minutes every half hour, I recommend not using this option if the selected time period is too short, it may not give you time to connect." *See, e.g.*, SAMSUNG_PRIORART0000361 (Purdy): "JuiceDefender lets you set a time interval—5 minutes, 15, 30, an hour, two hours— at which its background process will re-enable your carrier APN, see if there are new messages or data coming in, and then shut off again." *See, e.g.*, SAMSUNG_PRIORART0000335 (Ruddock):  **GreenPower App** POUZERATE0000196 (GreenPower User Guide) ("Global Wireless settings This is a shortcut to the phone system wireless settings where the user can find the setting "Mobile Network". That one should be checked or Green Power won't be able to properly manage Mobile Network. Screen off wireless delay This setting defines how long Green Power should wait before switching off wireless when the screen is turned off. Delaying turning off wireless is useful for instance if the user is reading something on the screen, not touching it. At some point the screen might turns off and you will touch it or press some buttons to switch it on again. Therefore, the wireless shouldn't be interrupted here. So, instead of switching off the wireless at once when the screen turns off, Green Power will wait that this delay elapses before switching off the wireless.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Wireless on delay This setting defines how long Green Power keeps the wireless on before turning it off again. This applies to the Wifi is the setting "Manage Wifi" is selected, and this applies to the Mobile Network if the setting "Manage Mobile Network" is selected. Wireless off delay This setting defines how long Green Power keeps the wireless off before turning it on again. This applies to the Wifi is the setting "Manage Wifi" is selected, and this applies to the Mobile Network if the setting "Manage Mobile Network" is selected. Screen on setting If this is selected, the wireless will be kept on when the screen is on. This applies to the Wifi is the setting "Manage Wifi" is selected, and this applies to the Mobile Network if the setting "Manage Mobile Network" is selected. If this is not selected, then Green Power will not make any difference whether the screen is on or off:: It will regularly switch on and off wireless if needed even if the screen is on. This can be useful if the you are using the phone for anything else than using wireless data (calling, playing local game, etc). In such a case you don't need the wireless to be always on. Power on setting If this is selected, the wireless will be kept on when the phone is connected to a power source. This applies to the Wifi is the setting "Manage Wifi" is selected, and this applies to the Mobile Network if the setting "Manage Mobile Network" is selected. This overrides the "Screen on setting": If this is selected and the power is connected, then wireless will be kept on whatever the screen state is. If this is not selected, then Green Power will not make any difference whether the phone is connected to the power or not:: It will regularly switch on and off wireless if needed.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

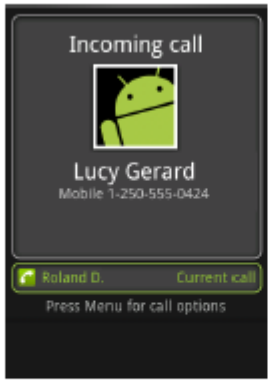
'578 Claims	Android Device with One or More Apps
	Check Traffic If this is selected, then prior to turning off wireless, Green Power will check that there is no network traffic. If there is, it will wait a few seconds and checks again until there is no traffic anymore.") SAMSUNG_PRIORART0005260 **Answer incoming, hold call in progress** The incoming call will display in the foreground.  Press CALL to answer the incoming call and place the call in progress on hold. You can also press MENU and select Hold call in progress & answer. To switch calls at any time, press MENU and select Switch calls. SAMSUNG_PRIORART0005207-09

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Data synchronization Some Google applications on your phone (Gmail, Calendar, and Contacts) give you access to the same personal information that you can add, view, and edit on your computer using Google Web applications. This means that when you add, change, or delete your information in any of these applications on the Web, the updated information also appears on your phone, and vice versa. Also, if you lose your phone or if your phone is destroyed, your personal information is not lost and will appear, as before, on a replacement phone. This mirroring of information happens through over-the-air data synchronization, or data "syncing". Data syncing occurs in the background and shouldn't ever get in your way. You'll know your data is being synchronized when you see this icon in the status bar: . Because sending large amounts of data back and forth over the air can take time and require considerable bandwidth, there are some settings on the phone that allow you to control data sync.

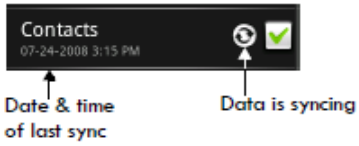
Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

Sync by application

To control synchronization for Gmail, Calendar, and Contacts, do the following:

1 Press HOME, then press MENU and select Settings.

2 Select Data synchronization.
When any of the applications are synchronizing, you will see the "sync" icon: . You will also see the last time your data was synchronized.



3 By default, the personal information in Gmail, Calendar, and Contacts will sync whenever you make a change or receive a new message. You can change this behavior:

Auto-sync	When selected, Auto-sync will sync Gmail, Calendar, and Contacts automatically, as you make changes or receive new messages. When OFF, information will not be synced automatically, although you can force a sync by using the individual application check boxes described below.
Gmail	Clear this check box to exclude Gmail from auto-sync. To force a sync, either clear the check box then select it, or press MENU and select Sync now. To control sync by Gmail label, you must do so from the Gmail settings screen. Read more in "Select labels to synchronize" on page 46.
Calendar	Clear this check box to exclude Calendar from auto-sync. To force a sync, either clear the check box then select it, or press MENU and select Sync now.
Contacts	Clear this check box to exclude Contacts from auto-sync. To force a sync, either clear the check box then select it, or press MENU and select Sync now.
Cancel sync	During a sync you can stop it by pressing MENU and selecting Cancel sync.
Sync problems	If you see this icon to the left of the sync check box, then there was a temporary problem with the data synchronization. Check your data connection and try again later.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Sync by Gmail label You can select to sync only Gmail messages with certain labels from the Gmail application settings screen. Read more in "Select labels to synchronize" on page 46. SAMSUNG_PRIORART0005245 Open in background — Select if you want links to new pages to open in a new window in the background. SAMSUNG_PRIORART0005598

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Android 1.0 **SAMSUNG_PRIORART0005487, Activity** * Activities in the system are managed as an activity stack. * When a new activity is started, it is placed on the top of the stack * and becomes the running activity -- the previous activity always remains * below it in the stack, and will not come to the foreground again until * the new activity exits. * * An activity has essentially four states: * * - If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * - If an activity has lost focus but is still visible (that is, a new non-full-sized * or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme * low memory situations. * - If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden * and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* completely restarted and restored to its previous state. * * * The following diagram shows the important state paths of an Activity. * The square rectangles represent callback methods you can implement to * perform operations when the Activity moves between states. The colored * ovals are major states the Activity can be in. * * <img src="../../../images/activity_lifecycle.png" * The entire lifetime of an activity happens between the first call * to {@link android.app.Activity#onCreate} through to a single final call * to {@link android.app.Activity#onDestroy}. An activity will do all setup * of "global" state in onCreate(), and release all remaining resources in * onDestroy(). For example, if it has a thread running in the background * to download data from the network, it may create that thread in onCreate() * and then stop the thread in onDestroy(). * * The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* * The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new * intent is delivered -- so the code in these methods should be fairly * lightweight. * * * The entire lifecycle of an activity is defined by the following * Activity methods. All of these are hooks that you can override * to do appropriate work when the activity changes state. All * activities will implement {@link android.app.Activity#onCreate} * to do their initial setup; many will also implement * {@link android.app.Activity#onPause} to commit changes to data and * otherwise prepare to stop interacting with the user. You should always * call up to your superclass when implementing these methods. * <tr><th colspan="2" align="left" border="0">{@link android.app.Activity#onStop onStop()}</th> * <td>Called when the activity is no longer visible to the user, because * another activity has been resumed and is covering this one. This * may happen either because a new activity is being started, an existing * one is being brought in front of this one, or this one is being * destroyed. * Followed by either <code>onRestart()</code> if * this activity is coming back to interact with the user, or

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* <code>onDestroy()</code> if this activity is going away.</td> * <td align="center">Yes</td> * <td align="center"><code>onRestart()</code> or * <code>onDestroy()</code></td> * </tr> * <p>The Android system attempts to keep application process around for as * long as possible, but eventually will need to remove old processes when * memory runs low. As described in Activity * Lifecycle, the decision about which process to remove is intimately * tied to the state of the user's interaction with it. In general, there * are four states a process can be in based on the activities running in it, * listed here in order of importance. The system will kill less important * processes (the last ones) before it resorts to killing more important * processes (the first ones). * * * <p>The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has * reached a memory paging state, so this is required in order to keep the user * interface responsive. * <p>A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * <p>A background activity (an activity that is not visible to

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its * {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it. * <p>An empty process is one hosting no activities or other * application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * /** * Called as part of the activity lifecycle when an activity is going into * the background, but has not (yet) been killed. The counterpart to * {@link #onResume}. * * <p>When activity B is launched in front of activity A, this callback will * be invoked on A. B will not be created until A's {@link #onPause} returns, * so be sure to not do anything lengthy here. * * <p>This callback is mostly used for saving any persistent state the * activity is editing, to present a "edit in place" model to the user and * making sure nothing is lost if there are not enough resources to start

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* the new activity without first killing this one. This is also a good * place to do things like stop animations and other things that consume a * noticeable mount of CPU in order to make the switch to the next activity * as fast as possible, or to close resources that are exclusive access * such as the camera. * * In situations where the system needs more memory it may kill paused * processes to reclaim resources. Because of this, you should be sure * that all of your state is saved by the time you return from * this function. In general {@link #onSaveInstanceState} is used to save * per-instance state in the activity and this method is used to store * global persistent data (in content providers, files, etc.) * * After receiving this call you will usually receive a following call * to {@link #onStop} (after the next activity has been resumed and * displayed), however in some cases there will be a direct call back to * {@link #onResume} without going through the stopped state. * * Derived classes must call through to the super class's * implementation of this method. If they do not, an exception will be * thrown. * * @see #onResume * @see #onSaveInstanceState * @see #onStop */ **SAMSUNG_PRIORART0005487, ActivityManager** /**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* Return a list of the tasks that are currently running, with
	* the most recent being first and older ones after in order. Note that
	* "running" does not mean any of the task's code is currently loaded or
	* activity -- the task may have been frozen by the system, so that it
	* can be restarted in its previous state when next brought to the
	* foreground.
	*
	* @param maxNum The maximum number of entries to return in the list. The
	* actual number returned may be smaller, depending on how many tasks the
	* user has started.
	*
	* @return Returns a list of RunningTaskInfo records describing each of
	* the running tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RunningTaskInfo> getRunningTasks(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningTaskInfo>)ActivityManagerNative.getDefault()
	.getTasks(maxNum, 0, null);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Set to true if the service has asked to run as a foreground process.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	*/ public boolean foreground; **Android 1.6** **SAMSUNG_PRIORART0005350, Fundamentals.jd** **Activities** An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others. Each one is implemented as a subclass of the {@link android.app.Activity} base class. An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one. Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen. The visual content of the window is provided by a hierarchy of views — objects derived from the base {@link android.view.View} class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions

'578 Claims	Android Device with One or More Apps
	directed at that space. Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more. A view hierarchy is placed within an activity's window by the {@link android.app.Activity#setContentView Activity.setContentView()} method. The *content view* is the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.) **Services** A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the {@link android.app.Service} base class. A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback. Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback). See Processes and Threads, later. All the activities in a task move together as a unit. The entire task (the entire activity stack) can be brought to the foreground or sent to the background. Suppose, for instance, that the current task has four

'578 Claims	**Android Device with One or More Apps**
	activities in its stack — three under the current activity. The user presses the HOME key, goes to the application launcher, and selects a new application (actually, a new *task*). The current task goes into the background and the root activity for the new task is displayed. Then, after a short period, the user goes back to the home screen and again selects the previous application (the previous task). That task, with all four activities in the stack, comes forward. When the user presses the BACK key, the screen does not display the activity the user just left (the root activity of the previous task). Rather, the activity on the top of the stack is removed and the previous activity in the same task is displayed. As noted above, there's never more than one instance of a "{@code singleTask}" or "{@code singleInstance}" activity, so that instance is expected to handle all new intents. A "{@code singleInstance}" activity is always at the top of the stack (since it is the only activity in the task), so it is always in position to handle the intent. However, a "{@code singleTask}" activity may or may not have other activities above it in the stack. If it does, it is not in position to handle the intent, and the intent is dropped. (Even though the intent is dropped, its arrival would have caused the task to come to the foreground, where it would remain.) An activity has essentially three states: • It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions. • It is *paused* if it has lost focus but is still visible to the user. That is, another activity lies on top of it and that activity either is transparent or doesn't cover the full screen, so some of the paused activity can show through. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations. • It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere. Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	• The **entire lifetime** of an activity happens between the first call to {@link android.app.Activity#onCreate onCreate()} through to a single final call to {@link android.app.Activity#onDestroy}. An activity does all its initial setup of "global" state in {@code onCreate()}, and releases all remaining resources in {@code onDestroy()}. For example, if it has a thread running in the background to download data from the network, it may create that thread in {@code onCreate()} and then stop the thread in {@code onDestroy()}. • The **visible lifetime** of an activity happens between a call to {@link android.app.Activity#onStart onStart()} until a corresponding call to {@link android.app.Activity#onStop onStop()}. During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. For example, you can register a {@link android.content.BroadcastReceiver} in {@code onStart()} to monitor for changes that impact your UI, and unregister it in {@code onStop()} when the user can no longer see what you are displaying. The {@code onStart()} and {@code onStop()} methods can be called multiple times, as the activity alternates between being visible and hidden to the user. • The **foreground lifetime** of an activity happens between a call to {@link android.app.Activity#onResume onResume()} until a corresponding call to {@link android.app.Activity#onPause onPause()}. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the resumed and paused states — for example, {@code onPause()} is called when the device goes to sleep or when a new activity is started, {@code onResume()} is called when an activity result or a new intent is delivered. Therefore, the code in these two methods should be fairly lightweight. `{@link android.app.Activity#onStart onStart()}` — Called just before the activity becomes visible to the user. Followed by {@code onResume()} if the activity comes to the foreground, or {@code onStop()} if it becomes hidden. — No — {@code onResume()} or {@code onStop()} Processes and lifecycles

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
	The Android system tries to maintain an application process for as long as possible, but eventually it will need to remove old processes when memory runs low. To determine which processes to keep and which to kill, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components. Processes with the lowest importance are eliminated first, then those with the next lowest, and so on. There are five levels in the hierarchy. The following list presents them in order of importance: 1. A **foreground process** is one that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions hold: ○ It is running an activity that the user is interacting with (the Activity object's {@link android.app.Activity#onResume onResume()} method has been called). ○ It hosts a service that's bound to the activity that the user is interacting with. ○ It has a {@link android.app.Service} object that's executing one of its lifecycle callbacks ({@link android.app.Service#onCreate onCreate()}, {@link android.app.Service#onStart onStart()}, or {@link android.app.Service#onDestroy onDestroy()}). ○ It has a {@link android.content.BroadcastReceiver} object that's executing its {@link android.content.BroadcastReceiver#onReceive onReceive()} method. Only a few foreground processes will exist at any given time. They are killed only as a last resort — if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive. 2. A **visible process** is one that doesn't have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions holds: ○ It hosts an activity that is not in the foreground, but is still visible to the user (its {@link android.app.Activity#onPause onPause()} method has been called). This may occur, for example, if the foreground activity is a dialog that allows the previous activity to be seen behind it. ○ It hosts a service that's bound to a visible activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	3. A **service process** is one that is running a service that has been started with the {@link android.content.Context#startService startService()} method and that does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing an mp3 in the background or downloading data on the network), so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes. 4. A **background process** is one holding an activity that's not currently visible to the user (the Activity object's {@link android.app.Activity#onStop onStop()} method has been called). These processes have no direct impact on the user experience, and can be killed at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and captures its current state, killing its process will not have a deleterious effect on the user experience. 5. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. **SAMSUNG_PRIORART0005487, Activity** * An activity has essentially four states: * * If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * If an activity has lost focus but is still visible (that is, a new non-full-sized * or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* low memory situations. * - If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden * and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be * completely restarted and restored to its previous state. * - The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user. * * - The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* intent is delivered -- so the code in these methods should be fairly * lightweight. * * Here is an excerpt from a calendar activity that stores the user's * preferred view mode in its persistent settings: * * <pre class="prettyprint"> * public class CalendarActivity extends Activity { * ... * * static final int DAY_VIEW_MODE = 0; * static final int WEEK_VIEW_MODE = 1; * The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has * reached a memory paging state, so this is required in order to keep the user * interface responsive. * A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * A background activity (an activity that is not visible to * the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its

'578 Claims	Android Device with One or More Apps
	* {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it. * <p>An empty process is one hosting no activities or other * application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * * <p>Sometimes an Activity may need to do a long-running operation that exists * independently of the activity lifecycle itself. An example may be a camera * application that allows you to upload a picture to a web site. The upload * may take a long time, and the application should allow the user to leave * the application will it is executing. To accomplish this, your Activity * should start a {@link Service} in which the upload takes place. This allows * the system to properly prioritize your process (considering it to be more * important than other non-visible applications) for the duration of the * upload, independent of whether the original activity is paused, stopped, * or finished. */ **SAMSUNG_PRIORART0005350, ActivityManager** /** * Return a list of the tasks that the user has recently launched, with * the most recent being first and older ones after in order. *

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* @param maxNum The maximum number of entries to return in the list. The
	* actual number returned may be smaller, depending on how many tasks the
	* user has started and the maximum number the system can remember.
	*
	* @return Returns a list of RecentTaskInfo records describing each of
	* the recent tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RecentTaskInfo> getRecentTasks(int maxNum, int flags)
	throws SecurityException {
	try {
	return ActivityManagerNative.getDefault().getRecentTasks(maxNum,
	flags);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Return a list of the tasks that are currently running, with
	* the most recent being first and older ones after in order. Note that
	* "running" does not mean any of the task's code is currently loaded or
	* activity -- the task may have been frozen by the system, so that it
	* can be restarted in its previous state when next brought to the
	* foreground.
	*
	* @param maxNum The maximum number of entries to return in the list. The

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* actual number returned may be smaller, depending on how many tasks the
	* user has started.
	*
	* @return Returns a list of RunningTaskInfo records describing each of
	* the running tasks.
	*
	* @throws SecurityException Throws SecurityException if the caller does
	* not hold the {@link android.Manifest.permission#GET_TASKS} permission.
	*/
	public List<RunningTaskInfo> getRunningTasks(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningTaskInfo>)ActivityManagerNative.getDefault()
	.getTasks(maxNum, 0, null);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Set to true if the service has asked to run as a foreground process.
	*/
	public boolean foreground;
	
	/**
	* Return a list of the services that are currently running.
	*
	* @param maxNum The maximum number of entries to return in the list. The

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* actual number returned may be smaller, depending on how many services
	* are running.
	*
	* @return Returns a list of RunningServiceInfo records describing each of
	* the running tasks.
	*/
	public List<RunningServiceInfo> getRunningServices(int maxNum)
	throws SecurityException {
	try {
	return (List<RunningServiceInfo>)ActivityManagerNative.getDefault()
	.getServices(maxNum, 0);
	} catch (RemoteException e) {
	// System dead, we will be dead too soon!
	return null;
	}
	}
	
	/**
	* Constant for {@link #importance}: this process is running the
	* foreground UI.
	*/
	public static final int IMPORTANCE_FOREGROUND = 100;
	
	/**
	* Constant for {@link #importance}: this process is running something
	* that is considered to be actively visible to the user.
	*/
	public static final int IMPORTANCE_VISIBLE = 200;
	
	/**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* Constant for {@link #importance}: this process is contains services * that should remain running. */ public static final int IMPORTANCE_SERVICE = 300; /** * Constant for {@link #importance}: this process process contains * background code that is expendable. */ public static final int IMPORTANCE_BACKGROUND = 400; /** * The relative importance level that the system places on this * process. May be one of {@link #IMPORTANCE_FOREGROUND}, * {@link #IMPORTANCE_VISIBLE}, {@link #IMPORTANCE_SERVICE}, * {@link #IMPORTANCE_BACKGROUND}, or {@link #IMPORTANCE_EMPTY}. These * constants are numbered so that "more important" values are always * smaller than "less important" values. */ public int importance; /** * An additional ordering within a particular {@link #importance} * category, providing finer-grained information about the relative * utility of processes within a category. This number means nothing * except that a smaller values are more recently used (and thus * more important). Currently an LRU value is only maintained for * the {@link #IMPORTANCE_BACKGROUND} category, though others may * be maintained in the future. */

'578 Claims	Android Device with One or More Apps

	public int lru;
	
	public RunningAppProcessInfo() {
	importance = IMPORTANCE_FOREGROUND;
	}
	
	/**
	* Returns a list of application processes that are running on the device.
	*
	* @return Returns a list of RunningAppProcessInfo records, or null if there are no
	* running processes (it will not return an empty list). This list ordering is not
	* specified.
	*/
	public List<RunningAppProcessInfo> getRunningAppProcesses() {
	try {
	return ActivityManagerNative.getDefault().getRunningAppProcesses();
	} catch (RemoteException e) {
	return null;
	}
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityManager
	
	/**
	* Class that answers queries about the state of network connectivity. It also
	* notifies applications when network connectivity changes. Get an instance
	* of this class by calling
	* {@link android.content.Context#getSystemService(String)
	Context.getSystemService(Context.CONNECTIVITY_SERVICE)}.
	* <p>

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* The primary responsibilities of this class are to:
	*
	* - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.)
	* - Send broadcast intents when network connectivity changes
	* - Attempt to "fail over" to another network when connectivity to a network
	* is lost
	* - Provide an API that allows applications to query the coarse-grained or fine-grained
	* state of the available networks
	*
	*/
	
	@SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
	public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED =
	"android.net.conn.BACKGROUND_DATA_SETTING_CHANGED";
	
	public static final int TYPE_MOBILE = 0;
	public static final int TYPE_WIFI = 1;
	
	public static final int DEFAULT_NETWORK_PREFERENCE = TYPE_WIFI;
	
	static public boolean isNetworkTypeValid(int networkType) {
	return networkType == TYPE_WIFI || networkType == TYPE_MOBILE;
	}
	
	public void setNetworkPreference(int preference) {
	try {
	mService.setNetworkPreference(preference);
	} catch (RemoteException e) {
	}
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/** {@hide} */
	public boolean setRadio(int networkType, boolean turnOn) {
	try {
	return mService.setRadio(networkType, turnOn);
	} catch (RemoteException e) {
	return false;
	}
	}
	
	/**
	* Returns the value of the setting for background data usage. If false,
	* applications should not use the network if the application is not in the
	* foreground. Developers should respect this setting, and check the value
	* of this before performing any background data operations.
	* <p>
	* All applications that have background services that use the network
	* should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}.
	*
	* @return Whether background data usage is allowed.
	*/
	public boolean getBackgroundDataSetting() {
	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	SAMSUNG_PRIORART0005350, ConnectivityService
	
	/*
	* Create the network state trackers for Wi-Fi and mobile
	* data. Maybe this could be done with a factory class,
	* but it's not clear that it's worth it, given that
	* the number of different network types is not going
	* to change very often.
	*/
	if (DBG) Log.v(TAG, "Starting Wifi Service.");
	mWifiStateTracker = new WifiStateTracker(context, handler);
	WifiService wifiService = new WifiService(context, mWifiStateTracker);
	ServiceManager.addService(Context.WIFI_SERVICE, wifiService);

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	mNetTrackers[ConnectivityManager.TYPE_WIFI] = mWifiStateTracker;
	
	mMobileDataStateTracker = new MobileDataStateTracker(context, handler);
	mNetTrackers[ConnectivityManager.TYPE_MOBILE] = mMobileDataStateTracker;
	
	/**
	* Make the state of network connectivity conform to the preference settings.
	* In this method, we only tear down a non-preferred network. Establishing
	* a connection to the preferred network is taken care of when we handle
	* the disconnect event from the non-preferred network
	* (see {@link #handleDisconnect(NetworkInfo)}).
	*/
	private void enforcePreference() {
	if (mActiveNetwork == null)
	return;
	
	for (NetworkStateTracker t : mNetTrackers) {
	if (t == mActiveNetwork) {
	int netType = t.getNetworkInfo().getType();
	int otherNetType = ((netType == ConnectivityManager.TYPE_WIFI) ?
	ConnectivityManager.TYPE_MOBILE :
	ConnectivityManager.TYPE_WIFI);
	
	if (t.getNetworkInfo().getType() != mNetworkPreference) {
	NetworkStateTracker otherTracker = mNetTrackers[otherNetType];
	if (otherTracker.isAvailable()) {
	teardown(t);
	}
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	}
	}
	}
	
	/**
	* @see ConnectivityManager#getBackgroundDataSetting()
	*/
	public boolean getBackgroundDataSetting() {
	return Settings.Secure.getInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, 1) == 1;
	}
	
	/**
	* @see ConnectivityManager#setBackgroundDataSetting(boolean)
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundDataUsage) {
	mContext.enforceCallingOrSelfPermission(
	android.Manifest.permission.CHANGE_BACKGROUND_DATA_SETTING,
	"ConnectivityService");
	
	if (getBackgroundDataSetting() == allowBackgroundDataUsage) return;
	
	Settings.Secure.putInt(mContext.getContentResolver(),
	Settings.Secure.BACKGROUND_DATA, allowBackgroundDataUsage ? 1 : 0);
	
	Intent broadcast = new Intent(
	ConnectivityManager.ACTION_BACKGROUND_DATA_SETTING_CHANGED);
	mContext.sendBroadcast(broadcast);
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	/**
	* See if the other network is available to fail over to.
	* If is not available, we enable it anyway, so that it
	* will be able to connect when it does become available,
	* but we report a total loss of connectivity rather than
	* report that we are attempting to fail over.
	*/
	NetworkInfo switchTo = null;
	if (newNet.isAvailable()) {
	mActiveNetwork = newNet;
	switchTo = newNet.getNetworkInfo();
	switchTo.setFailover(true);
	if (!switchTo.isConnectedOrConnecting()) {
	newNet.reconnect();
	}
	} else {
	newNet.reconnect();
	}
	
	if (info.getType() == ConnectivityManager.TYPE_MOBILE) {
	otherNet = mWifiStateTracker;
	} else /* info().getType() == TYPE_WIFI */ {
	otherNet = mMobileDataStateTracker;
	}
	
	int incrValue = ConnectivityManager.TYPE_MOBILE - ConnectivityManager.TYPE_WIFI;
	int stopValue = ConnectivityManager.TYPE_MOBILE + incrValue;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Android 2.2 **SAMSUNG_PRIORART0005350, Fundamentals.jd** **Activities** An *activity* presents a visual user interface for one focused endeavor the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others. Each one is implemented as a subclass of the {@link android.app.Activity} base class. An application might consist of just one activity or, like the text messaging application just mentioned, it may contain several. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one. Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows — for example, a pop-up dialog that calls for a user response in the midst of the activity, or a window that presents users with vital information when they select a particular item on-screen. The visual content of the window is provided by a hierarchy of views — objects derived from the base {@link android.view.View} class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space. Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more.

'578 Claims	Android Device with One or More Apps
	A view hierarchy is placed within an activity's window by the {@link android.app.Activity#setContentView Activity.setContentView()} method. The *content view* is the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.) **Services** A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it. Each service extends the {@link android.app.Service} base class. A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them. However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an ongoing service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback. Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback). See Processes and Threads, later. All the activities in a task move together as a unit. The entire task (the entire activity stack) can be brought to the foreground or sent to the background. Suppose, for instance, that the current task has four activities in its stack — three under the current activity. The user presses the HOME key, goes to the application launcher, and selects a new application (actually, a new *task*). The current task goes into the background and the root activity for the new task is displayed. Then, after a short period, the user goes back to the home screen and again selects the previous application (the previous task). That task, with all

'578 Claims	Android Device with One or More Apps
	four activities in the stack, comes forward. When the user presses the BACK key, the screen does not display the activity the user just left (the root activity of the previous task). Rather, the activity on the top of the stack is removed and the previous activity in the same task is displayed. As noted above, there's never more than one instance of a "{@code singleTask}" or "{@code singleInstance}" activity, so that instance is expected to handle all new intents. A "{@code singleInstance}" activity is always at the top of the stack (since it is the only activity in the task), so it is always in position to handle the intent. However, a "{@code singleTask}" activity may or may not have other activities above it in the stack. If it does, it is not in position to handle the intent, and the intent is dropped. (Even though the intent is dropped, its arrival would have caused the task to come to the foreground, where it would remain.) An activity has essentially three states: • It is *active* or *running* when it is in the foreground of the screen (at the top of the activity stack for the current task). This is the activity that is the focus for the user's actions. • It is *paused* if it has lost focus but is still visible to the user. That is, another activity lies on top of it and that activity either is transparent or doesn't cover the full screen, so some of the paused activity can show through. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations. • It is *stopped* if it is completely obscured by another activity. It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere. Taken together, these seven methods define the entire lifecycle of an activity. There are three nested loops that you can monitor by implementing them: • The **entire lifetime** of an activity happens between the first call to {@link android.app.Activity#onCreate onCreate()} through to a single final call to {@link android.app.Activity#onDestroy}. An activity does all its initial setup of "global" state in {@code onCreate()}, and releases all remaining resources in {@code onDestroy()}. For example, if it has

'578 Claims	Android Device with One or More Apps
	a thread running in the background to download data from the network, it may create that thread in {@code onCreate()} and then stop the thread in {@code onDestroy()}. • The **visible lifetime** of an activity happens between a call to {@link android.app.Activity#onStart onStart()} until a corresponding call to {@link android.app.Activity#onStop onStop()}. During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods, you can maintain resources that are needed to show the activity to the user. For example, you can register a {@link android.content.BroadcastReceiver} in {@code onStart()} to monitor for changes that impact your UI, and unregister it in {@code onStop()} when the user can no longer see what you are displaying. The {@code onStart()} and {@code onStop()} methods can be called multiple times, as the activity alternates between being visible and hidden to the user. • The **foreground lifetime** of an activity happens between a call to {@link android.app.Activity#onResume onResume()} until a corresponding call to {@link android.app.Activity#onPause onPause()}. During this time, the activity is in front of all other activities on screen and is interacting with the user. An activity can frequently transition between the resumed and paused states — for example, {@code onPause()} is called when the device goes to sleep or when a new activity is started, {@code onResume()} is called when an activity result or a new intent is delivered. Therefore, the code in these two methods should be fairly lightweight. {@link android.app.Activity#onStart onStart()} — Called just before the activity becomes visible to the user. Followed by {@code onResume()} if the activity comes to the foreground, or {@code onStop()} if it becomes hidden. — No — {@code onResume()} or {@code onStop()} Processes and lifecycles The Android system tries to maintain an application process for as long as possible, but eventually it will need to remove old processes when memory runs low. To determine which processes to keep and which to kill, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components. Processes with the lowest importance are eliminated first, then those

'578 Claims	Android Device with One or More Apps
	with the next lowest, and so on. There are five levels in the hierarchy. The following list presents them in order of importance: 6. A **foreground process** is one that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions hold: ○ It is running an activity that the user is interacting with (the Activity object's {@link android.app.Activity#onResume onResume()} method has been called). ○ It hosts a service that's bound to the activity that the user is interacting with. ○ It has a {@link android.app.Service} object that's executing one of its lifecycle callbacks ({@link android.app.Service#onCreate onCreate()}, {@link android.app.Service#onStart onStart()}, or {@link android.app.Service#onDestroy onDestroy()}). ○ It has a {@link android.content.BroadcastReceiver} object that's executing its {@link android.content.BroadcastReceiver#onReceive onReceive()} method. Only a few foreground processes will exist at any given time. They are killed only as a last resort — if memory is so low that they cannot all continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive. 7. A **visible process** is one that doesn't have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions holds: ○ It hosts an activity that is not in the foreground, but is still visible to the user (its {@link android.app.Activity#onPause onPause()} method has been called). This may occur, for example, if the foreground activity is a dialog that allows the previous activity to be seen behind it. ○ It hosts a service that's bound to a visible activity. A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. 8. A **service process** is one that is running a service that has been started with the {@link android.content.Context#startService startService()} method and that does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing an mp3 in the

'578 Claims	Android Device with One or More Apps
	background or downloading data on the network), so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes. 9. A **background process** is one holding an activity that's not currently visible to the user (the Activity object's {@link android.app.Activity#onStop onStop()} method has been called). These processes have no direct impact on the user experience, and can be killed at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and captures its current state, killing its process will not have a deleterious effect on the user experience. 10. An **empty process** is one that doesn't hold any active application components. The only reason to keep such a process around is as a cache to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches. **SAMSUNG_PRIORART0005487, Activity** * Activities in the system are managed as an activity stack. * When a new activity is started, it is placed on the top of the stack * and becomes the running activity -- the previous activity always remains * below it in the stack, and will not come to the foreground again until * the new activity exits. * * An activity has essentially four states: * * If an activity in the foreground of the screen (at the top of * the stack), * it is active or running. * If an activity has lost focus but is still visible (that is, a new non-full-sized

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* or transparent activity has focus on top of your activity), it * is paused. A paused activity is completely alive (it * maintains all state and member information and remains attached to * the window manager), but can be killed by the system in extreme * low memory situations. * - If an activity is completely obscured by another activity, * it is stopped. It still retains all state and member information, * however, it is no longer visible to the user so its window is hidden * and it will often be killed by the system when memory is needed * elsewhere. * - If an activity is paused or stopped, the system can drop the activity * from memory by either asking it to finish, or simply killing its * process. When it is displayed again to the user, it must be * completely restarted and restored to its previous state. * * * The following diagram shows the important state paths of an Activity. * The square rectangles represent callback methods you can implement to * perform operations when the Activity moves between states. The colored * ovals are major states the Activity can be in. * * * * There are three key loops you may be interested in monitoring within your * activity: * * * - The entire lifetime of an activity happens between the first call * to {@link android.app.Activity#onCreate} through to a single final call

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* to {@link android.app.Activity#onDestroy}. An activity will do all setup * of "global" state in onCreate(), and release all remaining resources in * onDestroy(). For example, if it has a thread running in the background * to download data from the network, it may create that thread in onCreate() * and then stop the thread in onDestroy(). * * The visible lifetime of an activity happens between a call to * {@link android.app.Activity#onStart} until a corresponding call to * {@link android.app.Activity#onStop}. During this time the user can see the * activity on-screen, though it may not be in the foreground and interacting * with the user. Between these two methods you can maintain resources that * are needed to show the activity to the user. For example, you can register * a {@link android.content.BroadcastReceiver} in onStart() to monitor for changes * that impact your UI, and unregister it in onStop() when the user an no * longer see what you are displaying. The onStart() and onStop() methods * can be called multiple times, as the activity becomes visible and hidden * to the user. * * The foreground lifetime of an activity happens between a call to * {@link android.app.Activity#onResume} until a corresponding call to * {@link android.app.Activity#onPause}. During this time the activity is * in front of all other activities and interacting with the user. An activity * can frequently go between the resumed and paused states -- for example when * the device goes to sleep, when an activity result is delivered, when a new * intent is delivered -- so the code in these methods should be fairly * lightweight. * *

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* <tr><th colspan="2" align="left" border="0">{@link android.app.Activity#onStart onStart()}</th> * <td>Called when the activity is becoming visible to the user. * Followed by <code>onResume()</code> if the activity comes * to the foreground, or <code>onStop()</code> if it becomes hidden.</td> * <td align="center">No</td> * <td align="center"><code>onResume()</code> or <code>onStop()</code></td> * </tr> * * <tr><td rowspan="2" style="border-left: none;"> </td> * <th align="left" border="0">{@link android.app.Activity#onResume onResume()}</th> * <td>Called when the activity will start * interacting with the user. At this point your activity is at * the top of the activity stack, with user input going to it. * Always followed by <code>onPause()</code>.</td> * <td align="center">No</td> * <td align="center"><code>onPause()</code></td> * </tr> * * * Unless you specify otherwise, a configuration change (such as a change * in screen orientation, language, input devices, etc) will cause your * current activity to be destroyed, going through the normal activity * lifecycle process of {@link #onPause}, * {@link #onStop}, and {@link #onDestroy} as appropriate. If the activity * had been in the foreground or visible to the user, once {@link #onDestroy} is * called in that instance then a new instance of the activity will be * created, with whatever savedInstanceState the previous instance had generated * from {@link #onSaveInstanceState}.

'578 Claims	Android Device with One or More Apps
	* <p>The foreground activity (the activity at the top of the screen * that the user is currently interacting with) is considered the most important. * Its process will only be killed as a last resort, if it uses more memory * than is available on the device. Generally at this point the device has * reached a memory paging state, so this is required in order to keep the user * interface responsive. * <p>A visible activity (an activity that is visible to the user * but not in the foreground, such as one sitting behind a foreground dialog) * is considered extremely important and will not be killed unless that is * required to keep the foreground activity running. * <p>A background activity (an activity that is not visible to * the user and has been paused) is no longer critical, so the system may * safely kill its process to reclaim memory for other foreground or * visible processes. If its process needs to be killed, when the user navigates * back to the activity (making it visible on the screen again), its * {@link #onCreate} method will be called with the savedInstanceState it had previously * supplied in {@link #onSaveInstanceState} so that it can restart itself in the same * state as the user last left it. * <p>An empty process is one hosting no activities or other * application components (such as {@link Service} or * {@link android.content.BroadcastReceiver} classes). These are killed very * quickly by the system as memory becomes low. For this reason, any * background operation you do outside of an activity must be executed in the * context of an activity BroadcastReceiver or Service to ensure that the system * knows it needs to keep your process around. * * * <p>Sometimes an Activity may need to do a long-running operation that exists * independently of the activity lifecycle itself. An example may be a camera

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* application that allows you to upload a picture to a web site. The upload
	* may take a long time, and the application should allow the user to leave
	* the application will it is executing. To accomplish this, your Activity
	* should start a {@link Service} in which the upload takes place. This allows
	* the system to properly prioritize your process (considering it to be more
	* important than other non-visible applications) for the duration of the
	* upload, independent of whether the original activity is paused, stopped,
	* or finished.
	*/
	
	/**
	* Called as part of the activity lifecycle when an activity is about to go
	* into the background as the result of user choice. For example, when the
	* user presses the Home key, {@link #onUserLeaveHint} will be called, but
	* when an incoming phone call causes the in-call Activity to be automatically
	* brought to the foreground, {@link #onUserLeaveHint} will not be called on
	* the activity being interrupted. In cases when it is invoked, this method
	* is called right before the activity's {@link #onPause} callback.
	*
	* <p>This callback and {@link #onUserInteraction} are intended to help
	* activities manage status bar notifications intelligently; specifically,
	* for helping activities determine the proper time to cancel a notfication.
	*
	* @see #onUserInteraction()
	*/
	protected void onUserLeaveHint() {
	}
	
	* <p>As a general rule, however, a resumed activity will have window
	* focus... unless it has displayed other dialogs or popups that take

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* input focus, in which case the activity itself will not have focus * when the other windows have it. Likewise, the system may display * system-level windows (such as the status bar notification panel or * a system alert) which will temporarily take window input focus without * pausing the foreground activity. * * @param hasFocus Whether the window of this activity has focus. * * @see #hasWindowFocus() /** * Change the desired orientation of this activity. If the activity * is currently in the foreground or otherwise impacting the screen * orientation, the screen will immediately be changed (possibly causing * the activity to be restarted). Otherwise, this will be used the next * time the activity is visible. * * @param requestedOrientation An orientation constant as used in * {@link ActivityInfo#screenOrientation ActivityInfo.screenOrientation}. */ public void setRequestedOrientation(int requestedOrientation) { **GOOG-HEADWATER-00000029, SAMSUNG_PRIORART0005353, ConnectivityManager** /** * Class that answers queries about the state of network connectivity. It also * notifies applications when network connectivity changes. Get an instance * of this class by calling

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* {@link android.content.Context#getSystemService(String) Context.getSystemService(Context.CONNECTIVITY_SERVICE)}. * <p> * The primary responsibilities of this class are to: * * - Monitor network connections (Wi-Fi, GPRS, UMTS, etc.) * - Send broadcast intents when network connectivity changes * - Attempt to "fail over" to another network when connectivity to a network * is lost * - Provide an API that allows applications to query the coarse-grained or fine-grained * state of the available networks * */ * A change in network connectivity has occurred. A connection has either * been established or lost. The NetworkInfo for the affected network is * sent as an extra; it should be consulted to see what kind of * connectivity event occurred. /** * Broadcast Action: The setting for **background data usage** has changed * values. Use {@link #getBackgroundDataSetting()} to get the current value. * <p> * If an application uses the network in the background, it should listen * for this broadcast and stop using the background data if the value is * false. */ @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION) public static final String ACTION_BACKGROUND_DATA_SETTING_CHANGED =

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	"android.net.conn.BACKGROUND_DATA_SETTING_CHANGED"; /** * The Default Mobile data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_MOBILE = 0; /** * The Default WIFI data connection. When active, all data traffic * will use this connection by default. Should not coexist with other * default connections. */ public static final int TYPE_WIFI = 1; /** * Returns the value of the setting **for background data usage**. If false, * applications should not use the network if the application is not in the * foreground. Developers should respect this setting, and check the value * of this before performing any background data operations. * <p> * All applications that have background services that use the network * should listen to {@link #ACTION_BACKGROUND_DATA_SETTING_CHANGED}. * * @return Whether background data usage is allowed. */ public boolean getBackgroundDataSetting() {

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**

	try {
	return mService.getBackgroundDataSetting();
	} catch (RemoteException e) {
	// Err on the side of safety
	return false;
	}
	}
	/**
	* Sets the value of the setting for background data usage.
	*
	* @param allowBackgroundData Whether an application should use data while
	* it is in the background.
	*
	* @attr ref android.Manifest.permission#CHANGE_BACKGROUND_DATA_SETTING
	* @see #getBackgroundDataSetting()
	* @hide
	*/
	public void setBackgroundDataSetting(boolean allowBackgroundData) {
	try {
	mService.setBackgroundDataSetting(allowBackgroundData);
	} catch (RemoteException e) {
	}
	}
	
	/**
	* Sets the persisted value for enabling/disabling Mobile data.
	*
	* @param enabled Whether the mobile data connection should be

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* used or not.
	* @hide
	*/
	public void setMobileDataEnabled(boolean enabled) {
	try {
	mService.setMobileDataEnabled(enabled);
	} catch (RemoteException e) {
	}
	}
	See also Android Developers Blog_Multitasking the Android Way, GOOG-HEADWATER-00000025-27
	
	A key to how Android handles applications in this way is that processes don't shut down cleanly. When the user leaves an application, its process is kept around in the background, allowing it to continue working (for example downloading web pages) if needed, and come immediately to the foreground if the user returns to it. If a device never runs out of memory, then Android will keep all of these processes around, truly leaving all applications "running" all of the time.
	
	Explicitly running in the background
	
	So far, we have a way for applications to implicitly do work in the background, as long as the process doesn't get killed by Android as part of its regular memory management. This is fine for things like loading web pages in the background, but what about features with harder requirements? Background music playback, data synchronization, location tracking, alarm clocks, etc.
	
	For these tasks, the application needs a way to tell Android "I would explicitly like to run at this point." There are two main facilities available to applications for this, represented by two kinds of components they can publish in their manifest: *broadcast receivers* and *services*.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Broadcast Receivers A BroadcastReceiver allows an application to run, for a brief amount of time, in the background as a result of something else happening. It can be used in many ways to build higher-level facilities: for example the AlarmManager allows an application to have a broadcast sent at a certain time in the future, and the LocationManager can send a broadcast when it detects interesting changes in location. Because information about the receiver is part of an application's manifest, Android can find and launch the application even if it isn't running; of course if it already has its process available in the background, the broadcast can very efficiently be directly dispatched to it. When handling a broadcast, the application is given a fixed set of time (currently 10 seconds) in which to do its work. If it doesn't complete in that time, the application is considered to be misbehaving, and its process immediately tossed into the background state to be killed for memory if needed. Broadcast receivers are great for doing small pieces of work in response to an external stimulus, such as posting a notification to the user after being sent a new GPS location report. They are very lightweight, since the application's process only needs to be around while actively receiving the broadcast. Because they are active for a deterministic amount of time, fairly strong guarantees can be made about not killing their process while running. However they are not appropriate for anything of indeterminate length, such as networking.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	Services A Service allows an application to implement longer-running background operations. There are actually a lot of other functions that services provide, but for the discussion here their fundamental purpose is for an application to say "hey I would like to continue running even while in the background, until I say I am done." An application controls when its service runs by explicitly starting and stopping the service. While services do provide a rich client-server model, its use is optional. Upon starting an application's services, Android simply instantiates the component in the application's process to provide its context. How it is used after that is up to the application: it can put all of the needed code inside of the service itself without interacting with other parts of the application, make calls on other singleton objects shared with other parts of the app, directly retrieve the Service instance from elsewhere if needed, or run it in another process and do a full-blown RPC protocol if that is desired. Process management for services is different than broadcast receivers, because an unbounded number of services can ask to be running for an unknown amount of time. There may not be enough RAM to have all of the requesting services run, so as a result no strong guarantees are made about being able to keep them running. If there is too little RAM, processes hosting services will be immediately killed like background processes are. However, if appropriate, Android will remember that these services wish to remain running, and restart their process at a later time when more RAM is available. For example, if the user goes to a web page that requires large amounts of RAM, Android may kill background service processes like sync until the browser's memory needs go down. Services can further negotiate this behavior by requesting they be considered "foreground." This places the service in a "please don't kill" state, but requires that it include a notification to the user about it actively running. This is useful for services such as background music playback or car navigation, which the user is actively aware of; when you're playing music and using the browser, you can always see the music-playing glyph in the status bar. Android won't try to kill these services, but as a trade-off, ensures the user knows about them and is able to explicitly stop them when desired.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	The value of generic components Android's generic broadcast receiver and service components allow developers to create a wide variety of efficient background operations, including things that were never originally considered. In Android 1.0 they were used to implement nearly all of the background behavior that the built-in and proprietary Google apps provided: • Music playback runs in a service to allow it to continue operating after the user leaves the music application. • The alarm clock schedules a broadcast receiver with the alarm manager, to go off at the next set alarm time.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	• The calendar application likewise schedules an alarm to display or update its notification at the appropriate time for the next calendar event. • Background file download is implemented a service that runs when there are any downloads to process. • The e-mail application schedules an alarm to wake up a service at regular intervals that looks for and retrieves any new mail. • The Google applications maintain a service to receive push notifications from the network; it in turn sends broadcasts to individual apps when it is told that they need to do things like synchronize contacts. As the platform has evolved, these same basic components have been used to implement many of the major new developer features: • Input methods are implemented by developers as a Service component that Android manages and works with to display as the current IME. • Application widgets are broadcast receivers that Android sends broadcasts to when it needs to interact with them. This allows app widgets to be quite lightweight, by not needing their application's process remain running. • Accessibility features are implemented as services that Android keeps running while in use and sends appropriate information to about user interactions. • Sync adapters introduced in Android 2.0 are services that are run in the background when a particular data sync needs to be performed. • Live wallpapers are a service started by Android when selected by the user.
[1e] one or more processors configured to implement an application program interface (API) that allows a particular application to access	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1d]. **Nexus One**

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
one or more aspects of the differential traffic control policy applicable to that application, including whether the user-settable aspects of the policy only allow the particular application to utilize the at least one WWAN for Internet service activities when the particular application is classified as interacting with a user in the device user interface foreground	*See, e.g.,* SAMSUNG_PRIORART0000001 (Nexus) at 319: **Development screen** The Development screen contains settings that are useful for developing Android applications. For full information, including documentation of the Android APIs and development tools, see the Android developer web site (http://developer.android.com). **USB debugging** Check to permit debugging tools on a computer to communicate with your phone via a USB connection. **Stay awake** Check to prevent the screen from dimming and locking when the phone is connected to a charger or to a USB device that provides power. Don't use this setting with a static image on the phone for long periods of time, or the screen may be marked with that image. **Allow mock locations** Check to permit a development tool on a computer to control where the phone believes it is located, rather than using the phone's own internal tools for this purpose.
[2] The wireless end-user device of claim 1, the one or more processors further configured to implement a network stack agent to apply the differential traffic control policy to Internet data service provided using the WWAN modem and the	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e]. In addition, *see, e.g.*: **Android 1.0** **SAMSUNG_PRIORART0005487, Socket.cpp** #include <utils/Socket.h> #include <utils/inet_address.h> #include <utils/Log.h>

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
at least one WWAN, such that an Internet service access request associated with at least one application that is not classified as interacting with a user in the device interface foreground is blocked, based at least on a user-settable aspect of the differential traffic control policy for that application.	#include <utils/Timers.h> #ifndef HAVE_WINSOCK # include <sys/types.h> # include <sys/socket.h> # include <netinet/in.h> # include <arpa/inet.h> #endif #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <string.h> #include <errno.h> #include <assert.h> using namespace android; /* * ========== * Socket * ========== */ #ifndef INVALID_SOCKET # define INVALID_SOCKET (-1) #endif #define UNDEF_SOCKET ((unsigned long) INVALID_SOCKET) /*static*/ bool Socket::mBootInitialized = false; /* * Extract system-dependent error code. */ static inline int getSocketError(void) {

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	return WSAGetLastError();
	#else
	return errno;
	#endif
	}
	/*
	* One-time initialization for socket code.
	*/
	/*static*/ bool Socket::bootInit(void)
	{
	#ifdef HAVE_WINSOCK
	WSADATA wsaData;
	int err;
	err = WSAStartup(MAKEWORD(2, 0), &wsaData);
	if (err != 0) {
	LOG(LOG_ERROR, "socket", "Unable to start WinSock\n");
	return false;
	}
	LOG(LOG_INFO, "socket", "Using WinSock v%d.%d\n",
	LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
	#endif
	mBootInitialized = true;
	return true;
	}
	/*
	* One-time shutdown for socket code.
	*/
	/*static*/ void Socket::finalShutdown(void)
	{

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	WSACleanup();
	#endif
	mBootInitialized = false;
	}
	/*
	* Simple constructor. Allow the application to create us and then make
	* bind/connect calls.
	*/
	Socket::Socket(void)
	: mSock(UNDEF_SOCKET)
	{
	if (!mBootInitialized)
	LOG(LOG_WARN, "socket", "WARNING: sockets not initialized\n");
	}
	/*
	* Destructor. Closes the socket and resets our storage.
	*/
	Socket::~Socket(void)
	{
	close();
	}
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const char* host, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	}
	InetSocketAddress sockAddr;
	if (!sockAddr.create(host, port))
	return -1;
	//return doConnect(sockAddr);
	int foo;
	foo = doConnect(sockAddr);
	return foo;
	}
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const InetAddress* addr, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	InetSocketAddress sockAddr;
	if (!sockAddr.create(addr, port))
	return -1;
	return doConnect(sockAddr);
	}
	/*
	* Finish creating a socket by connecting to the remote host.
	*
	* Returns 0 on success.
	*/
	int Socket::doConnect(const InetSocketAddress& sockAddr)
	{

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	#ifdef HAVE_WINSOCK
	SOCKET sock;
	#else
	int sock;
	#endif
	const InetAddress* addr = sockAddr.getAddress();
	int port = sockAddr.getPort();
	struct sockaddr_in inaddr;
	DurationTimer connectTimer;
	assert(sizeof(struct sockaddr_in) == addr->getAddressLength());
	memcpy(&inaddr, addr->getAddress(), addr->getAddressLength());
	inaddr.sin_port = htons(port);
	//fprintf(stderr, "--- connecting to %s:%d\n",
	// sockAddr.getHostName(), port);
	sock = ::socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (sock == INVALID_SOCKET) {
	int err = getSocketError();
	LOG(LOG_ERROR, "socket", "Unable to create socket (err=%d)\n", err);
	return (err != 0) ? err : -1;
	}
	connectTimer.start();
	if (::connect(sock, (struct sockaddr*) &inaddr, sizeof(inaddr)) != 0) {
	int err = getSocketError();
	LOG(LOG_WARN, "socket", "Connect to %s:%d failed: %d\n",
	sockAddr.getHostName(), port, err);
	return (err != 0) ? err : -1;
	}
	connectTimer.stop();
	if ((long) connectTimer.durationUsecs() > 100000) {
	LOG(LOG_INFO, "socket",

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	"Connect to %s:%d took %.3fs\n", sockAddr.getHostName(),
	port, ((long) connectTimer.durationUsecs()) / 1000000.0);
	}
	mSock = (unsigned long) sock;
	LOG(LOG_VERBOSE, "socket",
	"--- connected to %s:%d\n", sockAddr.getHostName(), port);
	return 0;
	}
	/*
	* Close the socket if it needs closing.
	*/
	bool Socket::close(void)
	{
	if (mSock != UNDEF_SOCKET) {
	//fprintf(stderr, "--- closing socket %lu\n", mSock);
	#ifdef HAVE_WINSOCK
	if (::closesocket((SOCKET) mSock) != 0)
	return false;
	#else
	if (::close((int) mSock) != 0)
	return false;
	#endif
	}
	mSock = UNDEF_SOCKET;
	return true;
	}
	/*
	* Read data from socket.
	*
	* Standard semantics: read up to "len" bytes into "buf". Returns the

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	* number of bytes read, or less than zero on error.
	*/
	int Socket::read(void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: read on invalid socket\n");
	return -500;
	}
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	cc = recv(sock, (char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}
	return cc;
	}
	/*
	* Write data to a socket.
	*
	* Standard semantics: write up to "len" bytes into "buf". Returns the
	* number of bytes written, or less than zero on error.
	*/
	int Socket::write(const void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {

'578 Claims	Android Device with One or More Apps

	LOG(LOG_ERROR, "socket", "ERROR: write on invalid socket\n");
	return -500;
	}
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	cc = send(sock, (const char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}
	return cc;
	
	Android 1.6
	
	SAMSUNG_PRIORART0005350, Socket.cpp
	
	//
	// Internet address class.
	//
	
	#ifdef HAVE_WINSOCK
	// This needs to come first, or Cygwin gets concerned about a potential
	// clash between WinSock and <sys/types.h>.
	# include <winsock2.h>
	#endif

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	#include <utils/Socket.h> #include <utils/inet_address.h> #include <utils/Log.h> #include <utils/Timers.h> #ifndef HAVE_WINSOCK # include <sys/types.h> # include <sys/socket.h> # include <netinet/in.h> # include <arpa/inet.h> #endif #include <stdlib.h> #include <stdio.h> #include <unistd.h> #include <string.h> #include <errno.h> #include <assert.h> using namespace android; /* * ========== * Socket * ========== */

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	#ifndef INVALID_SOCKET
	# define INVALID_SOCKET (-1)
	#endif
	#define UNDEF_SOCKET ((unsigned long) INVALID_SOCKET)
	
	/*static*/ bool Socket::mBootInitialized = false;
	
	/*
	* Extract system-dependent error code.
	*/
	static inline int getSocketError(void) {
	#ifdef HAVE_WINSOCK
	return WSAGetLastError();
	#else
	return errno;
	#endif
	}
	
	/*
	* One-time initialization for socket code.
	*/
	/*static*/ bool Socket::bootInit(void)
	{
	#ifdef HAVE_WINSOCK
	WSADATA wsaData;
	int err;
	
	err = WSAStartup(MAKEWORD(2, 0), &wsaData);
	if (err != 0) {

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	LOG(LOG_ERROR, "socket", "Unable to start WinSock\n");
	return false;
	}
	
	LOG(LOG_INFO, "socket", "Using WinSock v%d.%d\n",
	LOBYTE(wsaData.wVersion), HIBYTE(wsaData.wVersion));
	#endif
	
	mBootInitialized = true;
	return true;
	}
	
	/*
	* One-time shutdown for socket code.
	*/
	/*static*/ void Socket::finalShutdown(void)
	{
	#ifdef HAVE_WINSOCK
	WSACleanup();
	#endif
	mBootInitialized = false;
	}
	
	/*
	* Simple constructor. Allow the application to create us and then make
	* bind/connect calls.
	*/
	Socket::Socket(void)
	: mSock(UNDEF_SOCKET)

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	{
	if (!mBootInitialized)
	LOG(LOG_WARN, "socket", "WARNING: sockets not initialized\n");
	}
	
	/*
	* Destructor. Closes the socket and resets our storage.
	*/
	Socket::~Socket(void)
	{
	close();
	}
	
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const char* host, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	
	InetSocketAddress sockAddr;
	if (!sockAddr.create(host, port))
	return -1;
	
	//return doConnect(sockAddr);
	int foo;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	foo = doConnect(sockAddr);
	return foo;
	}
	
	/*
	* Create a socket and connect to the specified host and port.
	*/
	int Socket::connect(const InetAddress* addr, int port)
	{
	if (mSock != UNDEF_SOCKET) {
	LOG(LOG_WARN, "socket", "Socket already connected\n");
	return -1;
	}
	
	InetSocketAddress sockAddr;
	if (!sockAddr.create(addr, port))
	return -1;
	
	return doConnect(sockAddr);
	}
	
	/*
	* Finish creating a socket by connecting to the remote host.
	*
	* Returns 0 on success.
	*/
	int Socket::doConnect(const InetSocketAddress& sockAddr)
	{
	#ifdef HAVE_WINSOCK
	SOCKET sock;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	#else
	int sock;
	#endif
	const InetAddress* addr = sockAddr.getAddress();
	int port = sockAddr.getPort();
	struct sockaddr_in inaddr;
	DurationTimer connectTimer;
	
	assert(sizeof(struct sockaddr_in) == addr->getAddressLength());
	memcpy(&inaddr, addr->getAddress(), addr->getAddressLength());
	inaddr.sin_port = htons(port);
	
	//fprintf(stderr, "--- connecting to %s:%d\n",
	// sockAddr.getHostName(), port);
	
	sock = ::socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
	if (sock == INVALID_SOCKET) {
	int err = getSocketError();
	LOG(LOG_ERROR, "socket", "Unable to create socket (err=%d)\n", err);
	return (err != 0) ? err : -1;
	}
	
	connectTimer.start();
	
	if (::connect(sock, (struct sockaddr*) &inaddr, sizeof(inaddr)) != 0) {
	int err = getSocketError();
	LOG(LOG_WARN, "socket", "Connect to %s:%d failed: %d\n",
	sockAddr.getHostName(), port, err);
	return (err != 0) ? err : -1;
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	connectTimer.stop();
	if ((long) connectTimer.durationUsecs() > 100000) {
	LOG(LOG_INFO, "socket",
	"Connect to %s:%d took %.3fs\n", sockAddr.getHostName(),
	port, ((long) connectTimer.durationUsecs()) / 1000000.0);
	}
	
	mSock = (unsigned long) sock;
	LOG(LOG_VERBOSE, "socket",
	"--- connected to %s:%d\n", sockAddr.getHostName(), port);
	return 0;
	}
	
	/*
	* Close the socket if it needs closing.
	*/
	bool Socket::close(void)
	{
	if (mSock != UNDEF_SOCKET) {
	//fprintf(stderr, "--- closing socket %lu\n", mSock);
	#ifdef HAVE_WINSOCK
	if (::closesocket((SOCKET) mSock) != 0)
	return false;
	#else
	if (::close((int) mSock) != 0)
	return false;
	#endif
	}

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	mSock = UNDEF_SOCKET;
	
	return true;
	}
	
	/*
	* Read data from socket.
	*
	* Standard semantics: read up to "len" bytes into "buf". Returns the
	* number of bytes read, or less than zero on error.
	*/
	int Socket::read(void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: read on invalid socket\n");
	return -500;
	}
	
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	
	cc = recv(sock, (char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps

	}
	
	return cc;
	}
	
	/*
	* Write data to a socket.
	*
	* Standard semantics: write up to "len" bytes into "buf". Returns the
	* number of bytes written, or less than zero on error.
	*/
	int Socket::write(const void* buf, ssize_t len) const
	{
	if (mSock == UNDEF_SOCKET) {
	LOG(LOG_ERROR, "socket", "ERROR: write on invalid socket\n");
	return -500;
	}
	
	#ifdef HAVE_WINSOCK
	SOCKET sock = (SOCKET) mSock;
	#else
	int sock = (int) mSock;
	#endif
	int cc;
	
	cc = send(sock, (const char*)buf, len, 0);
	if (cc < 0) {
	int err = getSocketError();
	return (err > 0) ? -err : -1;
	}

'578 Claims	Android Device with One or More Apps
	return cc; } **Android 2.2** **SAMSUNG_PRIORART0005060, Socket** /** * Creates a new unconnected socket. When a SocketImplFactory is defined it * creates the internal socket implementation, otherwise the default socket * implementation will be used for this socket. * * @see SocketImplFactory * @see SocketImpl */ /** * Tries to connect a socket to all IP addresses of the given hostname. * * @param dstName * the target host name or IP address to connect to. * @param dstPort * the port on the target host to connect to. * @param localAddress * the address on the local host to bind to. * @param localPort * the port on the local host to bind to. * @param streaming

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* if {@code true} a streaming socket is returned, a datagram * socket otherwise. * @throws UnknownHostException * if the host name could not be resolved into an IP address. * @throws IOException * if an error occurs while creating the socket. * @throws SecurityException * if a security manager exists and it denies the permission to * connect to the given address and port. */ /** * Creates a new streaming socket connected to the target host specified by * the parameters {@code dstName} and {@code dstPort}. The socket is bound * to any available port on the local host. * <p>Implementation note: this implementation tries each * IP address for the given hostname until it either connects successfully * or it exhausts the set. It will try both IPv4 and IPv6 addresses in the * order specified by the system property {@code "java.net.preferIPv6Addresses"}. * * @param dstName * the target host name or IP address to connect to. * @param dstPort * the port on the target host to connect to. * @throws UnknownHostException * if the host name could not be resolved into an IP address. * @throws IOException * if an error occurs while creating the socket. * @throws SecurityException * if a security manager exists and it denies the permission to * connect to the given address and port.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	*/ /** * Checks whether the connection destination satisfies the security policy * and the validity of the port range. * * @param destAddr * the destination host address. * @param dstPort * the port on the destination host. */ /** * Closes the socket. It is not possible to reconnect or rebind to this * socket thereafter which means a new socket instance has to be created. * * @throws IOException * if an error occurs while closing the socket. */ /** * Gets the IP address of the target host this socket is connected to. * * @return the IP address of the connected target host or {@code null} if * this socket is not yet connected. */ /** * Gets the local IP address this socket is bound to. * * @return the local IP address of this socket or {@code InetAddress.ANY} if

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	Android Device with One or More Apps
	* the socket is unbound. */ /** * Gets the local port this socket is bound to. * * @return the local port of this socket or {@code -1} if the socket is * unbound. */ /** * Gets the port number of the target host this socket is connected to. * * @return the port number of the connected target host or {@code 0} if this * socket is not yet connected. */ /** * Gets the local address and port of this socket as a SocketAddress or * {@code null} if the socket is unbound. This is useful on multihomed * hosts. * * @return the bound local socket address and port. */ /** * Binds this socket to the given local host address and port specified by * the SocketAddress {@code localAddr}. If {@code localAddr} is set to * {@code null}, this socket will be bound to an available local address on * any free port.

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
	* * @param localAddr * the specific address and port on the local machine to bind to. * @throws IllegalArgumentException * if the given SocketAddress is invalid or not supported. * @throws IOException * if the socket is already bound or an error occurs while * binding. */ /** * Connects this socket to the given remote host address and port specified * by the SocketAddress {@code remoteAddr}. * * @param remoteAddr * the address and port of the remote host to connect to. * @throws IllegalArgumentException * if the given SocketAddress is invalid or not supported. * @throws IOException * if the socket is already connected or an error occurs while * connecting. */
[3] The wireless end-user device of claim 1, the API further to indicate, to the particular application,	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1e].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
one or more network access conditions based on the differential traffic control policy, wherein the one or network access conditions include a network access condition that indicates the unavailability to the particular application of an Internet data service that is currently available via the WWAN modem to a different application.	
[4] The wireless end-user device of claim 1, wherein the one or more processors are further configured to classify that the particular application is interacting with the user in the device user interface foreground when a user of the device is directly interacting with the	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].

'578 Claims	**Android Device with One or More Apps**
particular application or perceiving any benefit from the particular application.	
[5] The wireless end-user device of claim 1, wherein the one or more processors are further configured to classify that the particular application is interacting with the user in the device user interface foreground based on a state of user interface priority for the application.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].
[6] The wireless end-user device of claim 1, wherein the one or more processors are further configured to classify that the first end-user application is not interacting with the user in the device user interface foreground when the application is	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
providing or utilizing a background data service.	
[7] The wireless end-user device of claim 1, the user interface to provide a user of the device with information regarding why the differential traffic control policy is applied to the particular application.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e].
[8] The wireless end-user device of claim 1, wherein the differential traffic control policy is part of a multimode profile having different policies for different networks.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e].
[9] The wireless end-user device of claim 8, wherein the one or more processors are further configured to select a traffic control policy	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
from the multimode profile based at least in part on the type of network connection currently in use by the device.	
[10] The wireless end-user device of claim 9, wherein the one or more processors are further configured to, when the type of network connection is at least one type of WLAN connection, select a traffic control policy from the multimode profile based at least in part on a type of network connection from the WLAN to the Internet.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e].
[11] The wireless end-user device of claim 8, wherein the differential traffic control policy is the policy for a roaming WWAN network, the	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.,* the disclosures identified for claims [1 pre] - [1e].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
multimode profile having a second traffic control policy for a home WWAN network.	
[12] The wireless end-user device of claim 1, the one or more processors further comprising a network stack interface in communication with the API.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e], [2].
[13] The wireless end-user device of claim 1, further comprising a networking stack, wherein the one or more processors are further configured to, at an application service interface layer, identify application traffic flows prior to the flows entering the networking stack.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e], [2].
[14] The wireless end-user device of claim 1,	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures:

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
wherein the API comprises a network access API.	*See, e.g.*, the disclosures identified for claims[1e], [2].
[15] The wireless end-user device of claim 1, wherein the API further allows the particular application to access information indicating whether a current connected WWAN is a roaming network or a non-roaming network.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1e].
[16] The wireless end-user device of claim 1, wherein the API further informs the particular application when it is allowed to access Internet data service that is currently available via the WWAN modem.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1e].
[17] The wireless end-user device of claim 1, wherein the API informs the particular application of one or	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1e].

'578 Claims	**Android Device with One or More Apps**
more network traffic controls that the application is expected to implement.	
[18] The wireless end-user device of claim 1, wherein the API instructs the particular application to transition to a different state.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1e].
[19] The wireless end-user device of claim 1, wherein the one or more processors are configured to associate the particular application with the differential traffic control policy based on an application behavior.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].
[20] The wireless end-user device of claim 1, the API comprising a network stack interface that intercepts network socket interface messages for	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
applications and OS functions, the one or more processors configured to apply the differential traffic control policy to at least some of the intercepted network socket interface messages.	
[21] The wireless end-user device of claim 1, wherein the one or more processors are further configured to update the differential traffic control policy based on information received from a network element.	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e].
[22] The wireless end-user device of claim 1, wherein the one or more processors are configured to apply the differential traffic control policy to selectively block network access by the particular application by	Android Device with One or More Apps discloses and/or renders obvious this limitation. For example, see the following passages and/or figures, as well as related disclosures: *See, e.g.*, the disclosures identified for claims [1 pre] - [1e], [2].

Exhibit C-10 to Defendants' Amended Invalidity Contentions
U.S. Patent No. 9,521,578

'578 Claims	**Android Device with One or More Apps**
intercepting open, connect, and/or write requests by the particular application to a network stack.	